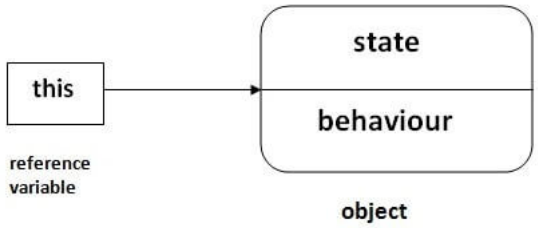
this keyword I

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



this keyword II

```
1  class Main
2  {
3        int instVar;
4
5        Main(int instVar)
6        {
7              this.instVar = instVar;
8              System.out.println("this reference = " + this);
9        }
10
11       public static void main(String[] args)
12       {
13             Main obj = new Main(8);
14             System.out.println("object reference = " + obj);
15       }
16 }
17 Output:
18 this reference = Main@23fc625e
19 object reference = Main@23fc625e
```

this keyword III

✓ There are various situations where this keyword is commonly used.

- 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 an argument in the constructor call.
- this can be used to return the current class instance from the method.

this keyword IV

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.

```
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);
      }
}
Output:
mc.age = 0
```

this keyword V

✓ The 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.

```
1  class Main
2  {
3
4        int age;
5        Main(int age)
6        {
7              this.age = age;
8        }
9
10       public static void main(String[] args)
11       {
12             Main obj = new Main(8);
13             System.out.println("obj.age = " + obj.age);
14       }
15 }
16 Output:
17 obj.age = 8
```

this keyword VI

Another example:

```
class Student
{
      int rollno;
      String name;
      float fee;
      Student(int rollno,String name,float fee)
      {
            rollno=rollno;
            name=name;
            fee=fee;
      }
      void display()
      {
            System.out.println(rollno+" "+name+" "+fee);
      }
}
class TestThis1
{
      public static void main(String args[])
      {
            Student s1=new Student(111,"ankit",5000f);
            Student s2=new Student(112,"sumit",6000f);
```

this keyword VII

```
23              s1.display();
24              s2.display();
25        }
26
27  }
28  Output:
29  0 null 0.0
30  0 null 0.0
```

✓ When this is used.

```
1   class Student
2   {
3         int rollno;
4         String name;
5         float fee;
6         Student(int rollno,String name,float fee)
7         {
8               this.rollno=rollno;
9               this.name=name;
10              this.fee=fee;
11        }
12        void display()
13        {
```

this keyword VIII

```
            System.out.println(rollno+" "+name+" "+fee);
      }
}
class TestThis2
{
      public static void main(String args[])
      {
            Student s1=new Student(111,"ankit",5000f);
            Student s2=new Student(112,"sumit",6000f);
            s1.display();
            s2.display();
      }

}
Output:
111 ankit 5000
112 sumit 6000
```

this with Getters and Setters:

```
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());
      }
}
Output: ???
```

this keyword X

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).

```
return_type method_name()
{
      return this;
}
```

this keyword XI

Example of this keyword that you return as a statement from the method:

```
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();
      }
}
```

this keyword XII

this() : to invoke current class constructor and in constructor overloading:

✓ 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.

```
1  //Calling default constructor from parameterized constructor:
2  class A
3  {
4        A()
5        {
6              System.out.println("hello a");
7        }
8        A(int x)
9        {
10             this();
11             System.out.println(x);
12       }
13 }
14 class TestThis5
15 {
16       public static void main(String args[])
17       {
18             A a=new A(10);
```

this keyword XIII

```
19        }
20
21  }
22  Output:
23  hello a
24  10
```

```
1   //Calling parameterized constructor from default constructor:
2   class A
3   {
4         A()
5         {
6               this(5);
7               System.out.println("hello a");
8         }
9         A(int x)
10        {
11              System.out.println(x);
12        }
13  }
14  public class TestThis6
15  {
16        public static void main(String args[])
17        {
```

this keyword XIV

```
            A a=new A();
      }

}
Output:
5
hello a
```

```
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
```

this keyword XV

```
17        private Complex()
18        {
19              // invokes the constructor with single parameter
20              this(0);
21        }
22        //Override
23        public String toString()
24        {
25              return this.a + " + " + this.b + "i";
26        }
27        public static void main( String[] args )
28        {
29
30              // creating object of Complex class
31              // calls the constructor with 2 parameters
32              Complex c1 = new Complex(2, 3);
33              // calls the constructor with a single parameter
34              Complex c2 = new Complex(3);
35              // calls the constructor with no parameters
36              Complex c3 = new Complex();
37              // print objects
38              System.out.println(c1);
39              System.out.println(c2);
```

this keyword XVI

```
            System.out.println(c3);
      }
}
Output:
2 + 3i
3 + 3i
0 + 0i
```

Passing this as an Argument:

```
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;
```

this keyword XVIII

```
23              o.y += 2;
24        }
25  }
26  public class Main
27  {
28        public static void main( String[] args )
29        {
30              ThisExample obj = new ThisExample(1, -2);
31        }
32  }
33  Output:
34  Before passing this to addTwo() method:
35  x = 1, y = -2
36  After passing this to addTwo() method:
37  x = 3, y = 0
```

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.

```
1  class A
2  {
3        void m()
4        {
5              System.out.println("hello m");
6        }
7        void n()
8        {
9              System.out.println("hello n");
10             //m(); //same as this.m()
11             this.m();
12       }
13 }
14 class TestThis4
15 {
16       public static void main(String args[])
17       {
18             A a=new A();
```

this keyword XX

```
19              a.n();
20          }
21
22  }
```

this keyword XXI

Real usage of this() constructor call:

✓ The this() constructor call should be used to reuse the constructor from the constructor. It maintains the chain between the constructors i.e. it is used for constructor chaining.

```
1  class Student
2  {
3        int rollno;
4        String name,course;
5        float fee;
6        Student(int rollno,String name,String course)
7        {
8              this.rollno=rollno;
9              this.name=name;
10             this.course=course;
11       }
12       Student(int rollno,String name,String course,float fee)
13       {
14             this(rollno,name,course); //reusing constructor. this() must be before this.fee; otherwise compiler will generate an error.
15             this.fee=fee;
16       }
17       void display()
18       {
```

this keyword XXII

```
            System.out.println(rollno+" "+name+" "+course+" "+fee);
      }
}
class TestThis7
{
      public static void main(String args[])
      {
            Student s1=new Student(111,"ankit","java");
            Student s2=new Student(112,"sumit","java",6000f);
            s1.display();
            s2.display();
      }

}
Output:
111 ankit java null
112 sumit java 6000
```

Program where this keyword is not required:

✓ If local variables(formal arguments) and instance variables are different, there is no need to use this keyword like in the following program:

```
1  class Student
2  {
3        int rollno;
4        String name;
5        float fee;
6        Student(int r,String n,float f)
7        {
8              rollno=r;
9              name=n;
10             fee=f;
11       }
12       void display()
13       {
14             System.out.println(rollno+" "+name+" "+fee);
15       }
16 }
17 class TestThis3
18 {
```

this keyword XXIV

```
19        public static void main(String args[])
20        {
21              Student s1=new Student(111,"ankit",5000f);
22              Student s2=new Student(112,"sumit",6000f);
23              s1.display();
24              s2.display();
25        }
26
27  }
28  Output:
29  111 ankit 5000
30  112 sumit 6000
```

super I

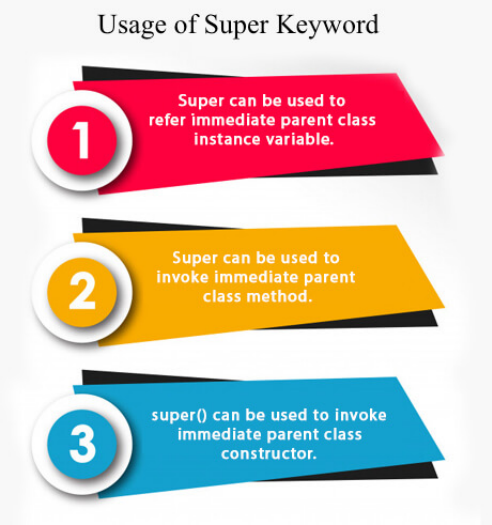
The super keyword in Java is a reference variable which is used to refer immediate parent class object.

✓ Whenever you create the instance of subclass, an instance of parent class is created implicitly which is referred by super reference variable.

Usage of Java super Keyword

1. super can be used to refer immediate parent class instance variable.
2. super can be used to invoke immediate parent class method.
3. super() can be used to invoke immediate parent class constructor.

super II


Usage of Super Keyword
1
Super can be used to refer immediate parent class instance variable.
2
Super can be used to invoke immediate parent class method.
3
super() can be used to invoke immediate parent class constructor.

super III

1. super is used to refer immediate parent class instance variable

```
1  class Animal
2  {
3        String color="white";
4  }
5  class Dog extends Animal
6  {
7        String color="black";
8        void printColor()
9        {
10             System.out.println(color); //prints color of Dog class
11             System.out.println(super.color); //prints color of Animal class
12       }
13 }
14 class TestSuper1
15 {
16       public static void main(String args[])
17       {
18             Dog d=new Dog();
19             d.printColor();
20       }
21
22 }
```

super IV

2. super can be used to invoke parent class method

```
class Animal
{
      void eat()
      {
            System.out.println("eating...");
      }
}
class Dog extends Animal
{
      void eat()
      {
            System.out.println("eating bread...");
      }
      void bark()
      {
            System.out.println("barking...");
      }
      void work()
      {
            super.eat();
            bark();
      }
```

super V

```
23 }
24 class TestSuper2
25 {
26       public static void main(String args[])
27       {
28             Dog d=new Dog();
29             d.work();
30       }
31
32 }
```

super VI

3. super is used to invoke parent class constructor.
   - ✓ The super keyword can also be used to invoke the parent class constructor.

```
1  class Animal
2  {
3        Animal()
4        {
5              System.out.println("animal is created");
6        }
7  }
8  class Dog extends Animal
9  {
10       Dog()
11       {
12             super();
13             System.out.println("dog is created");
14       }
15 }
16 class TestSuper3
17 {
18       public static void main(String args[])
19       {
20             Dog d=new Dog();
```

super VII

```
21          }
22
23  }
```

Note: super() is added in each class constructor automatically by compiler if there is no super() or this().

super example: real use

✓ Let's see the real use of super keyword. Here, Emp class inherits Person class so all the properties of Person will be inherited to Emp by default. To initialize all the property, we are using parent class constructor from child class. In such way, we are reusing the parent class constructor.

```
1  class Person
2  {
3        int id;
4        String name;
5        Person(int id,String name)
6        {
7              this.id=id;
8              this.name=name;
9        }
10 }
11 class Emp extends Person
12 {
13       float salary;
14       Emp(int id,String name,float salary)
```

super IX

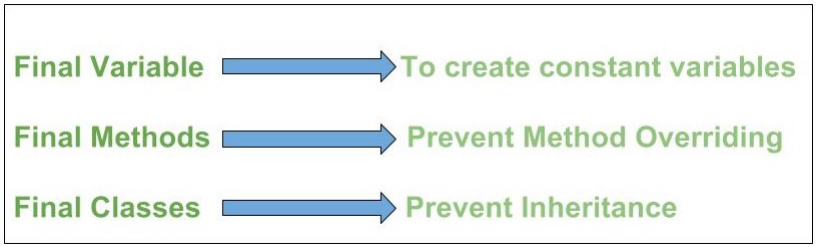
```
15        {
16              super(id,name); //reusing parent constructor
17              this.salary=salary;
18        }
19        void display()
20        {
21              System.out.println(id+" "+name+" "+salary);
22        }
23  }
24  class TestSuper5
25  {
26        public static void main(String[] args)
27        {
28              Emp e1=new Emp(1,"ankit",45000f);
29              e1.display();
30        }
31
32  }
```

Other Important points:

1. Call to super() must be first statement in Derived(Student) Class constructor.
2. If a constructor does not explicitly invoke a superclass constructor, the Java compiler automatically inserts a call to the no-argument constructor of the superclass.
3. If the superclass does not have a no-argument constructor, you will get a compile-time error. Object does have such a constructor, so if Object is the only superclass, there is no problem.
4. If a subclass constructor invokes a constructor of its superclass, either explicitly or implicitly, you might think that a whole chain of constructors called, all the way back to the constructor of Object.

final I

Final: Final keyword can be used with variable, method or class. It prevents from its content from being modified. When declared with class, it prevents the class from being extended.



final II

✓ When a variable is declared with final keyword, its value can't be modified, essentially, a constant.

✓ This also means that you must initialize a final variable.

✓ If the final variable is a reference, this means that the variable cannot be re-bound to reference another object, but internal state of the object pointed by that reference variable can be changed i.e. you can add or remove elements from final array or final collection.

✓ It is good practice to represent final variables in all uppercase, using underscore to separate words.

final III

```
1  // a final variable
2  final int THRESHOLD = 5;
3  // a blank final variable
4  final int THRESHOLD;
5  // a final static variable PI
6  static final double PI = 3.141592653589793;
7  // a blank final static variable
8  static final double PI;
```

Initializing a final variable :

✓ We must initialize a final variable, otherwise compiler will throw compile-time error.

✓ A final variable can only be initialized once, either via an initializer or an assignment statement.

There are three ways to initialize a final variable :

1. You can initialize a final variable when it is declared. This approach is the most common.

final IV

```
1  //Java program to demonstrate different ways of initializing a final variable
2  class Gfg
3  {
4        // a final variable direct initialize
5        final int THRESHOLD = 5;
6        // a blank final variable
7        final int CAPACITY;
8        // another blank final variable
9        final int MINIMUM;
10       // a final static variable PI direct initialize
11       static final double PI = 3.141592653589793;
12       // a blank final static variable
13       static final double EULERCONSTANT;
14       // instance initializer block for initializing CAPACITY
15       {
16             CAPACITY = 25;
17       }
18       // static initializer block for initializing EULERCONSTANT
19       static
20       {
21             EULERCONSTANT = 2.3;
22       }
23       // constructor for initializing MINIMUM
```

final V

```
24        // Note that if there are more than one constructor, you must initialize MINIMUM in them also
25        public GFG()
26        {
27              MINIMUM = -1;
28        }
29
30  }
```

final VI

2. A final variable is called blank final variable, if it is not initialized while declaration. A blank final variable can be initialized inside instance-initializer block or inside constructor.

```
1  class Demo
2  {
3        //Blank final variable
4        final int MAX_VALUE;
5
6        Demo()
7        {
8              //It must be initialized in constructor
9              MAX_VALUE=100;
10       }
11       void myMethod()
12       {
13             System.out.println(MAX_VALUE);
14       }
15       public static void main(String args[])
16       {
17             Demo obj=new Demo();
18             obj.myMethod();
```

final VII

```
19          }
20  }
```

3. If you have more than one constructor in your class then it must be initialized in all of them, otherwise compile time error will be thrown. A blank final static variable can be initialized inside static block.

final VIII

When to use a final variable :

✓ The only difference between a normal variable and a final variable is that we can re-assign value to a normal variable but we cannot change the value of a final variable once assigned.

✓ Hence final variables must be used only for the values that we want to remain constant throughout the execution of program.

Reference final variable :

✓ When a final variable is a reference to an object, then this final variable is called reference final variable.

```
1 final StringBuffer sb;
```

✓ Java program to demonstrate re-assigning final variable will throw compile-time error.

final IX

```java
// Java program to demonstrate re-assigning final variable will throw compile-time error
class Gfg
{
	static final int CAPACITY = 4;

	public static void main(String args[])
	{
		// re-assigning final variable
		// will throw compile-time error
		CAPACITY = 5;
	}
}
```

final X

✓ When a final variable is created inside a method/constructor/block, it is called local final variable, and it must initialize once where it is created. See below program for local final variable.

```
// Java program to demonstrate local final variable
// The following program compiles and runs fine
class Gfg
{
      public static void main(String args[])
      {
            // local final variable
            final int i;
            i = 20;
            System.out.println(i);
      }
}
```

final XI

✓ Note the difference between C++ const variables and Java final variables. const variables in C++ must be assigned a value when declared. For final variables in Java, it is not necessary as we see in above examples. A final variable can be assigned value later, but only once.

✓ final with foreach loop : final with for-each statement is a legal statement.

```
// Java program to demonstrate final
// with for-each statement

class Gfg
{
      public static void main(String[] args)
      {
            int arr[] =
            {
                  1, 2, 3
            }
            ;

            // final with for-each statement
            // legal statement
            for (final int i : arr)
```

final XII

```
17              System.out.print(i + " ");
18        }
19  }
20  Explanation : Since the i variable goes out of scope with each iteration of the loop, it is actually re-declaration each iteration,
          allowing the same token (i.e. i) to be used to represent multiple variables.
```

Whats the use of blank final variable?

✓ Lets say we have a Student class which is having a field called Roll No. Since Roll No should not be changed once the student is registered, we can declare it as a final variable in a class but we cannot initialize roll no in advance for all the students(otherwise all students would be having same roll no). In such case we can declare roll no variable as blank final and we initialize this value during object creation like this:

```
1  class StudentData
2  {
3        //Blank final variable
4        final int ROLL_NO;
5
6        StudentData(int rnum)
7        {
8              //It must be initialized in constructor
9              ROLL_NO=rnum;
10       }
11       void myMethod()
12       {
```

final XIV

```
13              System.out.println("Roll no is:"+ROLL_NO);
14        }
15        public static void main(String args[])
16        {
17              StudentData obj=new StudentData(1234);
18              obj.myMethod();
19        }
20  }
```

Uninitialized static final variable

✓ A static final variable that is not initialized during declaration can only be initialized in static block. Example:

```
1  class Example
2  {
3        //static blank final variable
4        static final int ROLL_NO;
5        static
6        {
7              ROLL_NO=1230;
8        }
9        public static void main(String args[])
10       {
11             System.out.println(Example.ROLL_NO);
12       }
13 }
```

Final methods:

When a method is declared with final keyword, it is called a final method.

✓ A final method cannot be overridden.

✓ The Object class does this—a number of its methods are final.

✓ We must declare methods with final keyword for which we required to follow the same implementation throughout all the derived classes.

```
1  class A
2  {
3        final void m1()
4        {
5              System.out.println("This is a final method.");
6        }
7  }
8
9  class B extends A
10 {
11       void m1()
12       {
```

final XVII

```
13              // COMPILE-ERROR! Can't override.
14              System.out.println("Illegal!");
15      }
16  }
```

✓ The above program would throw a compilation error, however we can use the parent class final method in sub class without any issues.

final XVIII

✓ This program would run fine as we are not overriding the final method. That shows that final methods are inherited but they are not eligible for overriding.

```
1  class XYZ
2  {
3        final void demo()
4        {
5              System.out.println("XYZ Class Method");
6        }
7  }
8  class ABC extends XYZ
9  {
10       public static void main(String args[])
11       {
12             ABC obj= new ABC();
13             obj.demo();
14       }
15 }
```

Final classes:

When a class is declared with final keyword, it is called a final class. A final class cannot be extended(inherited).

✓ There are two uses of a final class :

1. One is definitely to prevent inheritance, as final classes cannot be extended. For example, all Wrapper Classes like Integer,Float etc. are final classes. We can not extend them.

```
1 final class A
2 {
3       // methods and fields
4 }
5 // The following class is illegal.
6 class B extends A
7 {
8       // COMPILE-ERROR! Can't subclass A
9 }
```

2. The other use of final with classes is to create an immutable class like the predefined String class. You can not make a class immutable without making it final.

Points to Remember:

1. A constructor cannot be declared as final.
2. Local final variable must be initializing during declaration.
3. All variables declared in an interface are by default final.
4. We cannot change the value of a final variable.
5. A final method cannot be overridden.
6. A final class not be inherited.
7. If method parameters are declared final then the value of these parameters cannot be changed.

final XXI

3. It is a good practice to name final variable in all CAPS.
2. final, finally and finalize are three different terms. finally is used in exception handling and finalize is a method that is called by JVM during garbage collection.