final : Non-Access Modifier

In Java, a class, a field, a method and a variable can have "final" non-access modifier.

- **Final class:**
  When "final" non-access modifier is used for a class, that class can never be inherited i.e. that class can never have any subclasses.

  e.g.:

  Parent.java

```
public final class Parent
{
        public void fun()
        {
        System.out.println("This is inside the method 'fun'
        of the final class 'Parent'");
        }
}
```

  Child.java

```
public class Child extends Parent
{
        Public void gun()
        {
                System.out.println("This is inside the method
                'gun' of the final class 'Child'");

           }
      }
```

Output:

```
error: cannot inherit from final Parent
```

As here, 'Parent' has "final" non-access modifier, this class cannot have any subclasses and hence when we try to inherit the class 'Child' from final class 'Parent' it generates an error.

- **Final fields:**

1) **Non-static final fields:**

Non-static fields get memory when object of the class is created.
If we use "final" non-access modifier for non-static field, this fields gets behavior same as a constant variable in C or C++;
Final fields can only be initialized and cannot be assigned or reassigned!

e.g.:

Parent.java

```
      public class Parent
      {
           public final int a = 10;
```

```
            public static void main(String args[])
            {
                  Parent p = new Parent();
                  System.out.println("a = " + p.a);
            }
}
```

Output:

```
a = 10
```

In case a non-static field is not initialized, it can be assigned but only once in following ways as discussed below:

In constructor:

e.g.:

Parent.java

```
      public class Parent
      {
            public final int a;

            public Parent()
            {
                  a = 10;
            }

            public static void main(String args[])
            {
                  Parent p = new Parent();
```

```
				System.out.println("a = " + p.a);
			}
}
```

Output

```
a = 10
```

In non-static block:

e.g.:

Parent.java

```
	public class Parent
	{
		public final int a;

		{
			a = 10;
		}

		public static void main(String args[])
		{
			Parent p = new Parent();
			System.out.println("a = " + p.a);
		}
}
```

Output

```
a = 10
```

2) Static fields:

Static fields get memory before the object is created, hence they can be initialized or can be assigned once only in static block.

e.g.:

Parent.java

```
	public class Parent
	{
		public static final int a = 10;

		public static void main(String args[])
		{
			Parent p = new Parent();
			System.out.println("a = " + p.a);
		}
}
```

Output:

```
a = 10
```

e.g.:

Parent.java

```
	public class Parent
	{
		public static final int a;

	static
		{
```

```
                a = 10;
        }

        public static void main(String args[])
        {
                Parent p = new Parent();
                System.out.println("a = " + p.a);
        }
}
```

Output:

```
a = 10
```

Final methods:

If we don't want a method to be overridden, we can use "final" non-access modifier for such methods.

e.g.:

Parent.java

```
    public class Parent
    {
        public final void fun()
        {
            System.out.println("This in inside of final
    method 'fun' in class Parent");
        }
}
```

Child.java

```
	public class Child extends Parent
	{
			public void fun()
			{
				System.out.println("This in inside of final
	method 'fun' in class Child");
			}
}
```

Output

```
error: fun() in Child cannot override fun() in Parent
```

Final local variables:

Local variables get memory when the method is invoked i.e. gets called.

Final local variables can only be initialized or can be assigned but only once provided they're not initialized.

e.g.:

Parent.java

```
	public class Parent
	{
		public void fun()
		{
			final int a = 10;
			System.out.println("a = " + a);
		}
		public static void main(String args[])
		{
			Parent p = new Parent();
			p.fun();
		}
}
```

Output

```
a = 10
```