Hierarchy of abstraction in java

Anything whose context is not clear is said to be as abstract.

Let’s relate to the real world example, when you were a baby you didn’t know anything about how your future, career, schooling anything. But when you are 10 years old you’ll have some idea about what you want to be based on the influence of people you are surrounded with. But you still aren’t clear with what measures you take to achieve that and why you need to be that is still not clear. So many things are still abstract and not concrete yet. A decade passes by and now you are in 20’s and you might have completed your education, but still not clear with where you’ll be working, whom you’d marry etc. So still there are not some things which are abstract but the level of abstraction is reduced. And once you are 30, now you know where you are working, your designation and whom you’ll be married to. So generally now many things are concrete and abstraction is reduced.

Similarly in java also as you come down the hierarchy all methods and class are more concrete and less abstract.

Let’s try to understand with code

```
abstract class Bird
{
    abstract void eat();
    abstract void fly();
}
abstract class Eagle extends Bird
{
    void fly()
    {
        System.out.println("Eagle flies at great heights");
    }

}
class SerpentEagle extends Eagle
{
    void eat()
    {
        System.out.println("Serpent Eagle hunts over mountains and eats"
    }
}
```

```java
class SerpentEagle extends Eagle
{
    void eat()
    {
        System.out.println("Serpent Eagle hunts over mountains and eats");
    }
}
class GoldenEagle extends Eagle
{
    void eat()
    {
        System.out.println("Golden Eagle hunts over mountains and eats");
    }
}
class Demo
{
    public static void main(String[] args)
    {
        SerpentEagle se = new SerpentEagle();
        GoldenEagle ge = new GoldenEagle();

        se.eat();
        se.fly();
        ge.eat();
        ge.fly();
    }
}
```

Output:

```
Serpent Eagle hunts over mountains and eats
Eagle flies at great heights
Golden Eagle hunts over mountains and eats
Eagle flies at great heights
Press any key to continue . . .
```

Key Points

- **An abstract class can contain both abstract methods as well as concrete methods.**
- **A normal class can inherit from an abstract class.**
- **An abstract class can inherit from another abstract class.**
- **An abstract class can inherit from a normal class.**
- **An abstract class can contain only concrete methods as well.**

If a class contain even a single abstract method then the class should be declared as abstract.

Abstract class objects cannot be created/instantiated. If you want to create it then it must be inherited from another class.

Final keyword in java

The **final keyword** in java is used to restrict the user. The java final keyword can be used in many context. Final can be:

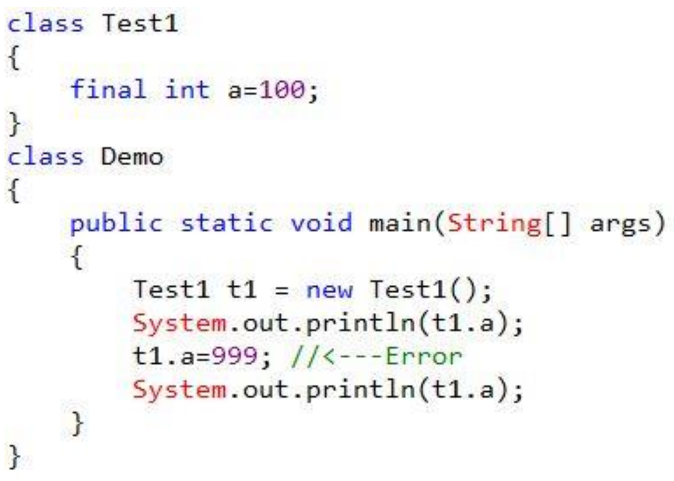
1. variable
2. method
3. class

The final keyword can be applied with the variables, **a final variable that have no value it is called blank final variable or uninitialized final variable**. It can be **initialized in the constructor only**. The blank final variable can be static also which will be initialized in the static block only. We will have detailed learning of these. Let's first learn the basics of final keyword.

Final variable

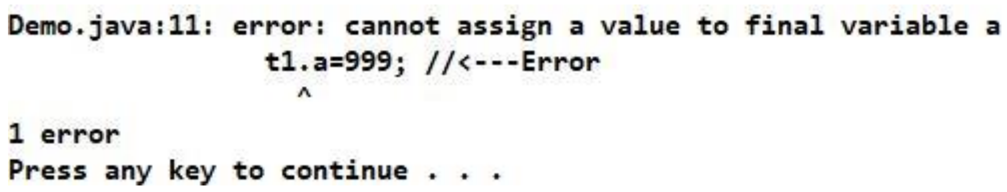
If you make any variable as final, you **cannot change the value** of final variable (It will be constant).

Example:



```
class Test1
{
    final int a=100;
}
class Demo
{
    public static void main(String[] args)
    {
        Test1 t1 = new Test1();
        System.out.println(t1.a);
        t1.a=999; //<---Error
        System.out.println(t1.a);
    }
}
```

Output:

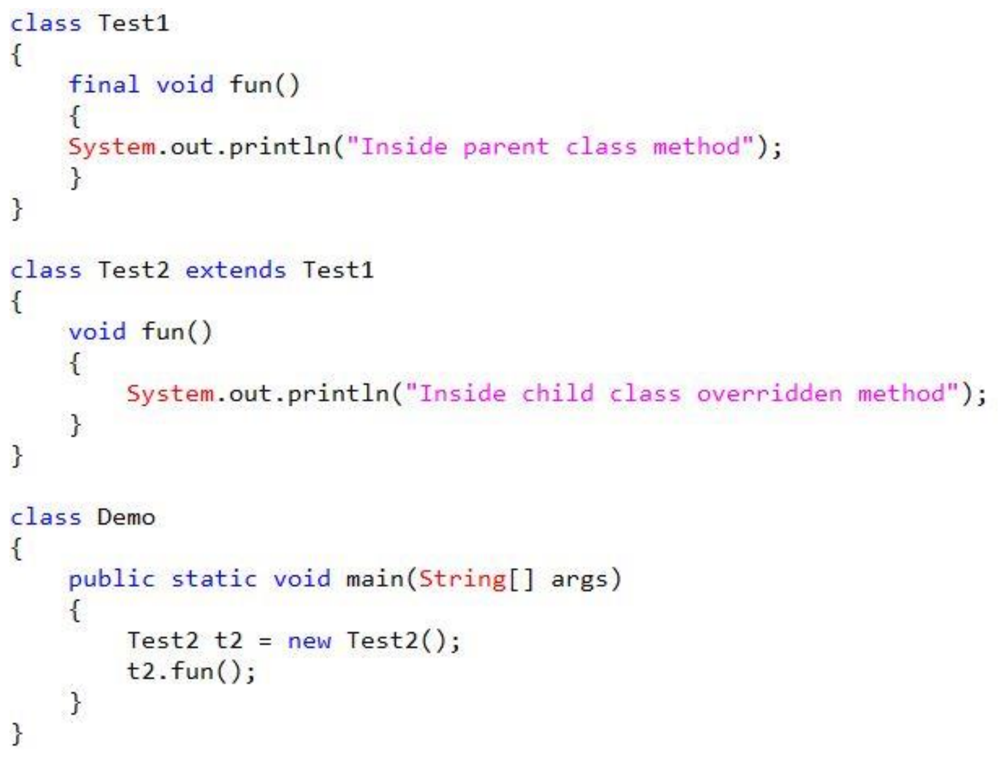


```
Demo.java:11: error: cannot assign a value to final variable a
                t1.a=999; //<---Error
                  ^
1 error
Press any key to continue . . .
```

final class

If you make any method as final, you **cannot override** it.

Example:

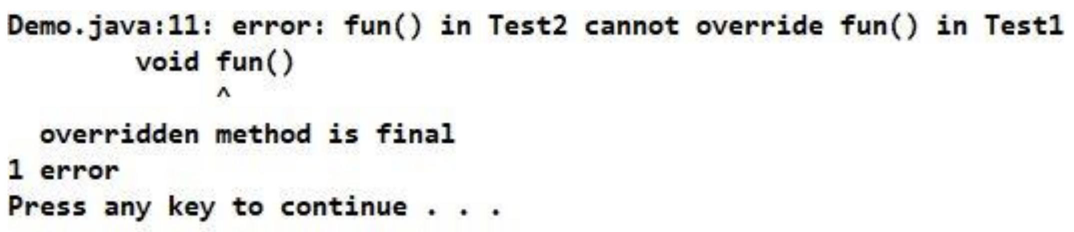


```
class Test1
{
     final void fun()
     {
     System.out.println("Inside parent class method");
     }
}

class Test2 extends Test1
{
     void fun()
     {
          System.out.println("Inside child class overridden method");
     }
}

class Demo
{
     public static void main(String[] args)
     {
          Test2 t2 = new Test2();
          t2.fun();
     }
}
```

Output:



```
Demo.java:11: error: fun() in Test2 cannot override fun() in Test1
        void fun()
             ^
  overridden method is final
1 error
Press any key to continue . . .
```

If you make any class as final, you **cannot extend** it.

Example:

```java
final class Test1
{
    void fun()
    {
    System.out.println("Inside parent class method");
    }
}

class Test2 extends Test1
{
    void fun()
    {
        System.out.println("Inside child class overridden method");
    }
}

class Demo
{
    public static void main(String[] args)
    {
        Test2 t2 = new Test2();
        t2.fun();
    }
}
```

Output:

```
Demo.java:9: error: cannot inherit from final Test1
class Test2 extends Test1
                    ^
1 error
Press any key to continue . . .
```