Access modifiers: Different types of keyword which decide whether the entity (class, method, variable etc) are accessible at a specific location or not.

Class level access modifier:

1. public

2. <default>

3. final

4. abstract

1. public : Whenever we declare a class as public then we can access that class throughout the project.

Example:

**public** **class** ACTest1 {

}

2. <default>: Whenever we declare a class as default then we can only access that class with in the package but not outside the package.

Example:

**class** ACTest3 {

}

3. final : It is a modifier which actually used with either public or default. Whenever we declare a class as final then we will not be able to create its child class.

**public** **final** **class** Test {

**public** **void** m1()

{

System.***out***.println("m1 method is executing");

}

}

**public** **class** Child **extends** Test {

**public** **static** **void** main(String[] args) {

Child c = **new** Child();

c.m1();

}

}

Note: In the above example we cannot create the child class of Test class as it is final in nature hence we will not be able to access its method.

4. abstract : It is an access modifier which is applicable to class as well as methods. Which provides functionality to represent complete as well as incomplete methods in a class.

1. Whenever we know some information partially and we have to represent that in terms of code then we can declare that class as abstract which can contain complete (concrete method) as well as incomplete methods(abstract method) .

2. We can have a class as abstract with no method in it or no incomplete complete methods in it. But if we have a method as abstract then we must have to declare that class as abstract.

3. We cannot create the object of an abstract class because abstract class can have complete as well as incomplete methods.

**Method / variable level access modifiers:**

1. public:

2. <default>

3. private

4. protected

1. public : When we declare a method / variable as public then we will be able to access that method / variable throughout the project but provided the class which contains the public method / variable has the accessibility to get access that location.

// public access modifier for method

**public** **static** **void** m1()

{

System.***out***.println("m1 method");

}

**public** **class** VariableAccessModifier {

**public** **int** i =10;

}

2. <default>: When we declare a method / variable as <default> then we will be able to access that method / variable with in the package only we cannot access that method / variable outside the package.

// default access modifier method

**void** m2()

{

System.***out***.println("m2 method from ACTest1 class");

}

**public** **class** VariableAccessModifier {

**static** String *s* = "abc";

}

3. private: When we declare a method / variable as private then we will only be able to access that method / variable within the class only we cannot access it outside the class through any means.

// private access modifier method

**private** **static** **void** m3()

{

System.***out***.println("m3 private method from ACTest1 class");

}

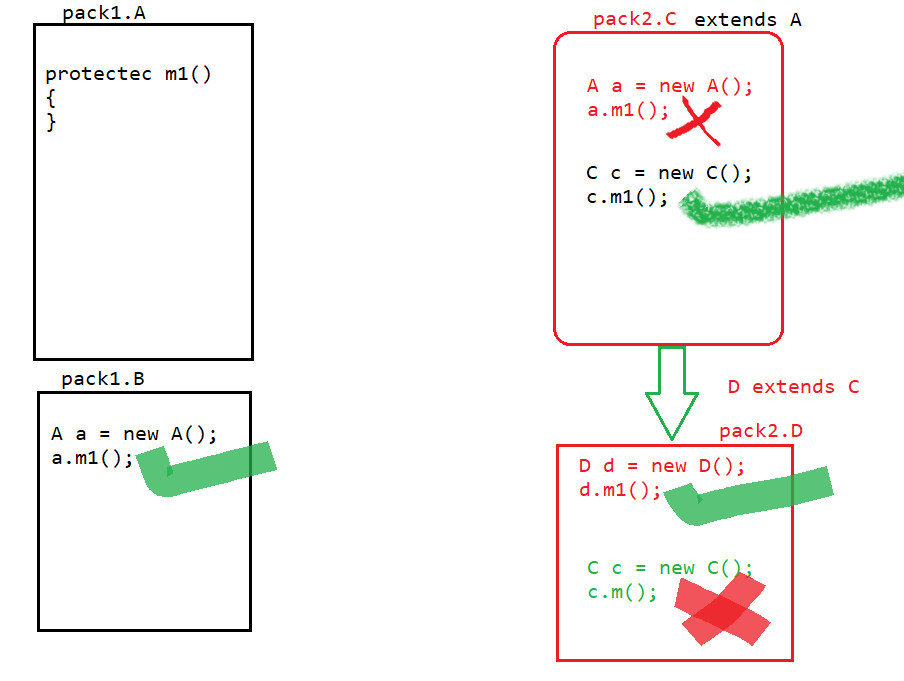
**public** **class** VariableAccessModifier {

**private** **static** **char** *c* = '$';

3. protected: When we declare a method / variable as a protected then with in the same package we can access it like <default> but for outside the package it can only be accessible inside the child class through the child reference variable only.

If the protected method / variable is static then we can access that method/ variable by using the static way i.e classname.methodname / variablename inside the child class.

Protected = <default> + reference variable of child class within the child for outside the package

example:

**public** **class** A {

**protected** **void** m1()

{

System.***out***.println("protected m1 method from A class");

}

}**package** protectedtestpackage;

**import** accessmodifiers.A;

**public** **class** B **extends** A

{

**public** **static** **void** main(String[] args) {

B b= **new** B();

b.m1();

A a = **new** A();

// a.m1();// we cannot access this method by A reference variable as it is protected which can only be access by reference variable of child class outside the package.

}

}

Static protected method example:

**public** **class** A {

**protected** **static** **void** m2()

{

System.***out***.println("protected static method A class");

}

**package** protectedtestpackage;

**import** accessmodifiers.A;

**public** **class** B **extends** A

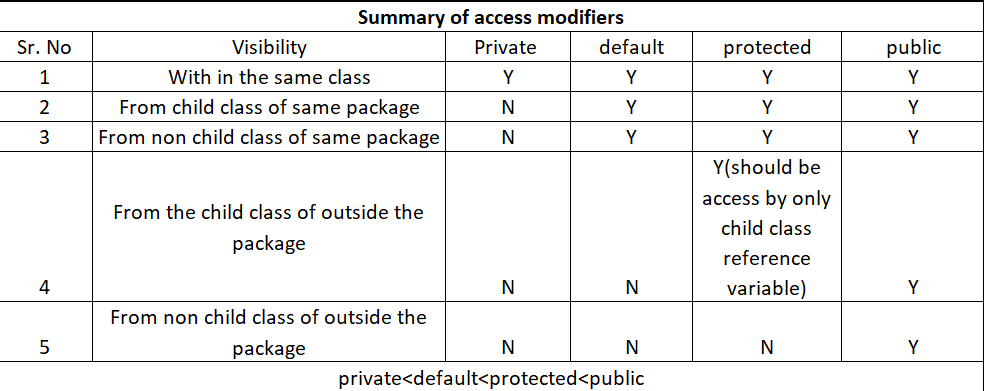
{

**public** **static** **void** main(String[] args) {

A.*m2*();

}

}



5. final: Whenever we declare a method as final then we cannot override that method but we can access that method based on the access modifier of that method**.**

**public** **class** Parent1 {

**public** **final** **void** marry() {

System.***out***.println("marry method of parent class");

}

**public** **class** Child1 **extends** Parent1 {

**public** **void** marry()

{

System.***out***.println("Child class marry method");

}

**public** **static** **void** main(String[] args) {

Child1 c = **new** Child1();

c.car();

c.home();

}

Note: In the above example the method marry is final inside the parent class hence it cannot be overridden inside the child class.

Final variable: If we declare a variable as final then we will be able to access that variable any where based on the access modifier but we cannot re- initialize its value except the location where we have declared it.

Example:

**public** **class** VariableAccessModifier {

**public** **final** **static** **int** ***j*** = 60;

**public** **static** **void** main(String[] args) {

System.***out***.println(*c*);

A.*m2*();

System.***out***.println(***j***);//60

}

**public** **void** m1()

{

System.***out***.println(***j***);

***j***= ***j***+2;// this line will give an error saying final cannot be reinitialize.

}