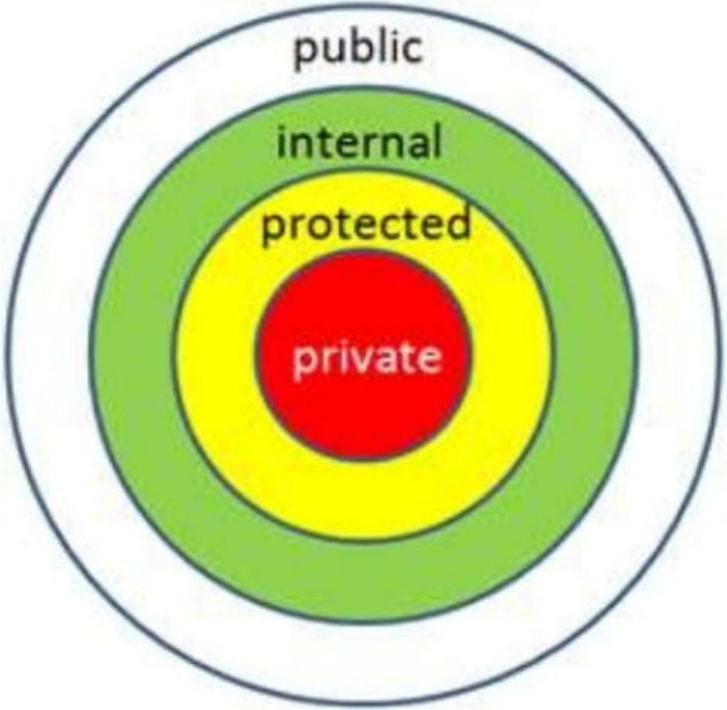
Access Modifiers

Syeda Ajbina Nusrat, Lecturer, CSE, UITS

Access Modifiers in java



Access Modifiers in java

- There are two types of modifiers in java: **access modifiers** and **non-access modifiers**.
- The access modifiers in java specifies accessibility (scope) of a data member, method, constructor or class.
- There are 4 types of java access modifiers:
  - private
  - default
  - protected
  - public
- There are many non-access modifiers such as static, abstract, synchronized, native, volatile, transient etc. Here, we will learn access modifiers.

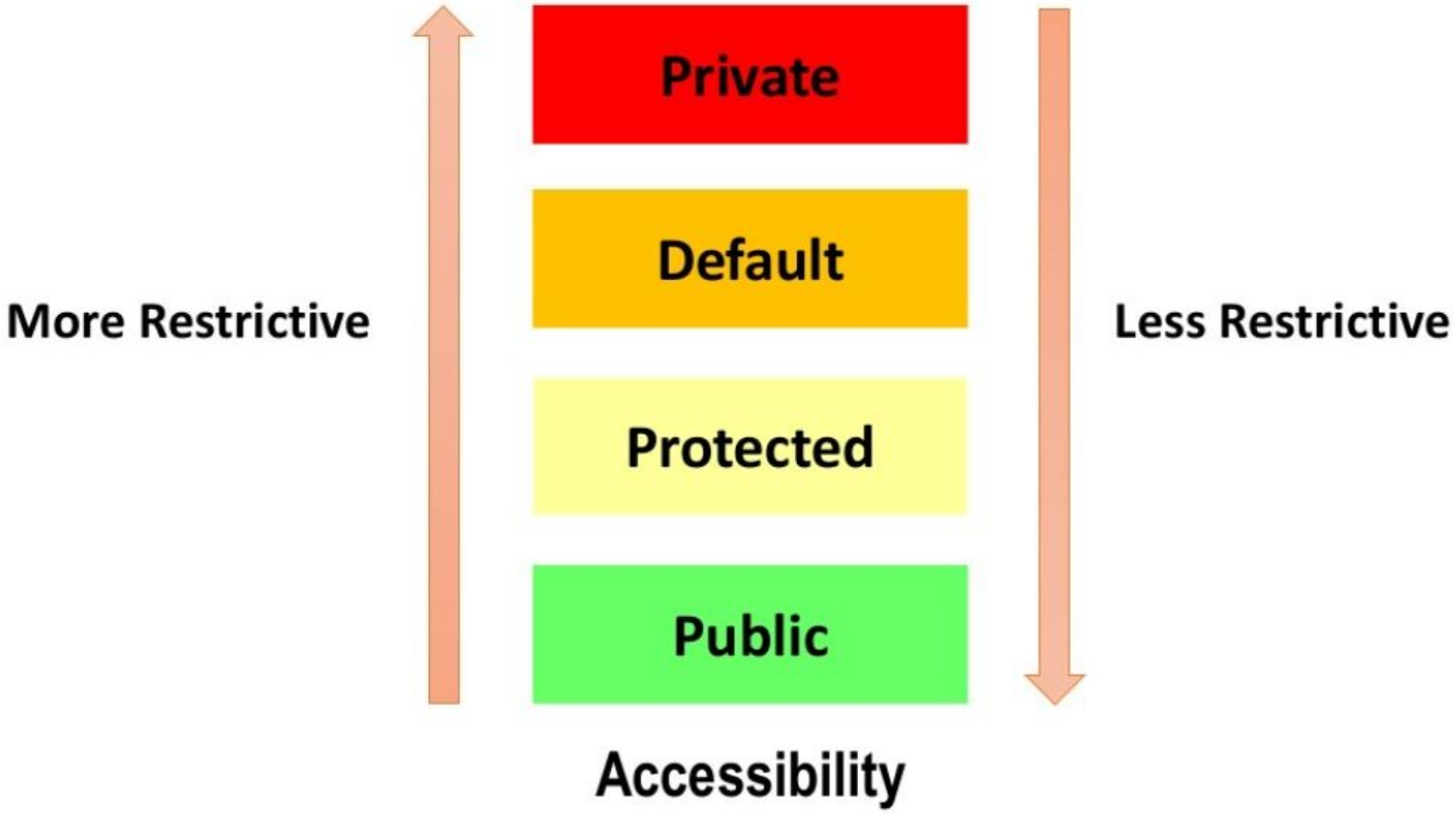
Access Modifiers in java

- **Access Control Modifiers**
- Java provides a number of access modifiers to set access levels for classes, variables, methods and constructors. The four access levels are –
  - Visible to the package, the default. No modifiers are needed.
  - Visible to the class only (private).
  - Visible to the world (public).
  - Visible to the package and all subclasses (protected).

Access Modifiers in java

- **Non-Access Modifiers**
- Java provides a number of non-access modifiers to achieve many other functionality.
  - The ***static*** modifier for creating class methods and variables.
  - The ***final*** modifier for finalizing the implementations of classes, methods, and variables.
  - The ***abstract*** modifier for creating abstract classes and methods.
  - The ***synchronized*** and ***volatile*** modifiers, which are used for threads.

Access Modifiers in java



Access Modifiers in java

- **1) private Access Modifier**
  - The private access modifier is accessible only within class.
  - Methods, variables, and constructors that are declared private can only be accessed within the declared class itself.
  - Private access modifier is the most restrictive access level. Class and interfaces cannot be private.
  - Variables that are declared private can be accessed outside the class, if public getter methods are present in the class.
  - Using the private modifier is the main way that an object encapsulates itself and hides data from the outside world.

Private Access Modifier

```
public class Test{
   private int data=40;
 public static void main(String args[]){
  Test ex = new Test();
  System.out.println("Data is: "+ex.data);


  }
}
```

Private Access Modifier

```
public class Test{
  private int data=40;
 public static void main(String args[]){
  Test ex = new Test();
  System.out.println("Data is: "+ex.data);


  }
}
```

Here the *data* variable of the Example class is private and this variable accessed from same class itself

Private Access Modifier

```
public class Test{
   private int data=40;
 public static void main(String args[]){
  Test ex = new Test();
  System.out.println("Data is: "+ex.data);

  }
}
```

Data variable is assess in the same class where it is defined

Private Access Modifier

```
public class Test{
   private int data=40;
 public static void main(String args[]){
  Test ex = new Test();
  System.out.println("Data is: "+ex.data);


  }
}
```

Output is:
Data is: 40

Private Access Modifier

```
public class Test{
   private int data;
 public static void main(String args[]){
   ex.data=40;
   Test ex = new Test();
   System.out.println("Data is: "+ex.data);

   }
}
```

Same Program as previous but single modification

Now we set the value of the variable here using the object of Example class

Output is:
Data is: 40

Private Access Modifier

```
class A{
   private int data=40;
   private void msg() {
   System.out.println("Hello java");}
   }
public class Test{
 public static void main(String args[]){
  A obj=new A();
  System.out.println(obj.data);  //Compile Time Error
  obj.msg();  //Compile Time Error
  }
}
```

Private Access Modifier

```
class A{
   private int data=40;
   private void msg() {
   System.out.println("Hello java");}
   }
public class Test{
 public static void main(String args[]){
  A obj=new A();
  System.out.println(obj.data);  //Compile Time Error
  obj.msg();  //Compile Time Error
  }
}
```

This is first class where we declare a private variable and define private method

Private Access Modifier

```
class A{
   private int data=40;
   private void msg() {
   System.out.println("Hello java");}
   }
public class Test{
 public static void main(String args[]){
  A obj=new A();
  System.out.println(obj.data);  //Compile Time Error
  obj.msg();  //Compile Time Error
  }
}
```

This is second class where we try to access private variable and method

Private Access Modifier

```
class A{
  private int data=40;
  private void msg() {
  System.out.println("Hello java");}
  }
public class Test{
 public static void main(String args[]){
  A obj=new A();
  System.out.println(obj.data);  //Compile Time Error
  obj.msg();  //Compile Time Error
  }
}
```

Instance variable is private here

Private Access Modifier

```
class A{
   private int data=40;
   private void msg() {
   System.out.println("Hello java");}
   }
public class Test{
 public static void main(String args[]){
  A obj=new A();
  System.out.println(obj.data);  //Compile Time Error
  obj.msg();  //Compile Time Error
  }
}
```

Private method is here

Private Access Modifier

```
class A{
   private int data=40;
   private void msg() {
   System.out.println("Hello java");}
   }
public class Test{
 public static void main(String args[]){
  A obj=new A();
  System.out.println(obj.data);  //Compile Time Error
  obj.msg();  //Compile Time Error
  }
}
```

Here, the data variable of the A class is private, so there's no way for other classes to retrieve or set its value directly.

Private Access Modifier

- So, if we want to make this variable available to the outside world, we defined two public methods:
  - *getter()*, which returns the value of variable, and
  - *setter(parameter)*, which sets its value of the variable.

```
class A {
  private int data;
  public int getA() {
    return this.data;
  }
  public void setA(int data) {
    this.data=data;
  }
}
```

Private Access Modifier

```
class A{
  private int data;
  public int getA() {
     return this.data;
   }
   public void setA(int data) {
     this.data=data;  }
}
public class Test{
 public static void main(String args[]){
   A obj=new A();
   obj.setA(12);
   System.out.println("Data is: "+obj.getA());
   }
}
```

Private Access Modifier

```
class A{
  private int data;
  public void setA(int data) {
    this.data=data;  }
  public int getA() {
    return this.data; }
}
public class Test{
 public static void main(String args[]){
  A obj=new A();
  obj.setA(12);
  System.out.println("Data is: "+obj.getA());
  }
}
```

Private instance variable

Private Access Modifier

```
class A{
  private int data;
  public void setA(int data) {
    this.data=data;  }
  public int getA() {
    return this.data; }
}
public class Test{
 public static void main(String args[]){
  A obj=new A();
  obj.setA(12);
  System.out.println("Data is: "+obj.getA());
  }
}
```

Public setter method

Private Access Modifier

```
class A{
  private int data;
  public void setA(int data) {
    this.data=data;  }
  public int getA() {
    return this.data; }
}
public class Test{
 public static void main(String args[]){
  A obj=new A();
  obj.setA(12);
  System.out.println("Data is: "+obj.getA());
  }
}
```

Public getter method

Private Access Modifier

```
class A{
  private int data;
  public void setA(int data) {
    this.data=data;  }
  public int getA() {
    return this.data; }
}
public class Test{
 public static void main(String args[]){
  A obj=new A();
  obj.setA(12);
  System.out.println("Data is: "+obj.getA());
  }
}
```

Here we set the value of the private variable

Private Access Modifier

```
class A{
  private int data;
  public void setA(int data) {
    this.data=data;  }
  public int getA() {
    return this.data; }
}
public class Test{
 public static void main(String args[]){
  A obj=new A();
  obj.setA(12);
  System.out.println("Data is: "+obj.getA());
  }
}
```

Here we get the value of the private variable

Private Access Modifier

```
class A{
  private int data;
  public void setA(int data) {
    this.data=data;  }
  public int getA() {
    return this.data; }
}
public class Test{
 public static void main(String args[]){
  A obj=new A();
  obj.setA(12);
  System.out.println("Data is: "+obj.getA());
  }
}
```

Accessing possible

Output is:
Data is: 12

Try to avoid using setter and getter methods
Because these methods make your data is publically

Role of Private Constructor

- The private modifier when applied to a constructor works in much the same way as when applied to a normal method or even an instance variable.
- Defining a constructor with the private modifier says that only the native class (as in the class in which the private constructor is defined) is allowed to create an instance of the class, and no other caller is permitted to do so.
- There are two possible reasons why one would want to use a private constructor – the first is that you don't want any objects of your class to be created at all, and the second is that you only want objects to be created internally – as in only created in your class.

Role of Private Constructor

- **1.Private constructors can be used in the singleton design pattern**
- Why would you want objects of your class to only be created internally?
- This could be done for any reason, but one possible reason is that you want to implement a singleton. A singleton is a design pattern that allows only one instance of your class to be created, and this can be accomplished by using a private constructor.

Role of Private Constructor

- **2.Private constructors can prevent creation of objects**
- The other possible reason for using a private constructor is to prevent object construction entirely. When would it make sense to do something like that? Of course, when creating an object doesn't make sense – and this occurs when the class only contains static members. And when a class contains only static members, those members can be accessed using only the class name – no instance of the class needs to be created.

Role of Private Constructor

- Java always provides a default, no-argument, public constructor if no programmer-defined constructor exists. Creating a private no-argument constructor essentially prevents the usage of that default constructor, thereby preventing a caller from creating an instance of the class. Note that the private constructor may even be empty.

- Let see an example in the next slide if we make any class constructor private, we cannot create the instance of that class from outside the class

Role of Private Constructor

```
class A{
   private A(){    //private constructor
   }
   void msg(){
   System.out.println("Hello java");}
   }
public class Test{
 public static void main(String args[]){
  A obj=new A();    //Compile Time Error
 }
}
```

Access Modifiers in java

- **2) public Access Modifier**
  - The **public access modifier** is accessible everywhere. It has the widest scope among all other modifiers.
  - A class, method, constructor, interface, etc. declared public can be accessed from any other class. Therefore, fields, methods, blocks declared inside a public class can be accessed from any class belonging to the Java Universe.
  - However, if the public class we are trying to access is in a different package, then the public class still needs to be imported. Because of class inheritance, all public methods and variables of a class are inherited by its subclasses.

Note: About package we learn later in detail.. ☺

public Access Modifier

```
class A{
  public int data=40;
}
public class Test{
 public static void main(String args[]){
  A obj=new A();
  System.out.println("Data is: "+obj.data);
  }
}
```

public Access Modifier

```
class A{
  public int data=40;
}
public class Test{
 public static void main(String args[]){
  A obj=new A();
  System.out.println("Data is: "+obj.data);
  }
}
```

public instance variable and accessible in other classes

public Access Modifier

```
class A{
  public int data=40;
}
public class Test{
 public static void main(String args[]){
  A obj=new A();
  System.out.println("Data is: "+obj.data);
  }
}
```

Output is:
Data is: 40

Access Modifiers in java

- **3) protected access modifier**
  - The **protected access modifier** is accessible within package and outside the package but through inheritance only.
  - The protected access modifier can be applied on the data member, method and constructor. It can't be applied on the class.
  - Protected access gives the subclass a chance to use the helper method or variable, while preventing a nonrelated class from trying to use it.

protected Access Modifier

```
class A{
  protected int data=40;
}
public class Test{
 public static void main(String args[]){
  A obj=new A();
  System.out.println("Data is: "+obj.data);
  }
}
```

protected Access Modifier

```
class A{
  protected int data=40;
}
public class Test{
 public static void main(String args[]){
  A obj=new A();
  System.out.println("Data is: "+obj.data);
  }
}
```

Protected instance variable

protected Access Modifier

```
class A{
  protected int data=40;
}
public class Test{
 public static void main(String args[]){
  A obj=new A();
  System.out.println("Data is: "+obj.data);
  }
}
```

Access protected instance variable in other class

Output is:
Data is: 40

protected Access Modifier

```
class AudioPlayer {
  protected boolean openSpeaker(Speaker sp) {
    // implementation details
  }
}
class StreamingAudioPlayer {
  boolean openSpeaker(Speaker sp) {
    // implementation details
  }
}
```

Here, if we define openSpeaker() method as private, then it would not be accessible from any other class other than AudioPlayer. If we define it as public, then it would become accessible to all the outside world. But our intention is to expose this method to its subclass only, that's why we have used protected modifier.

Access Modifiers in java

- **4) default access modifier**
  - Default access modifier means we do not explicitly declare an access modifier for a class, field, method, etc.
  - A variable or method declared without any access control modifier is available to any other class in the same package. The fields in an interface are implicitly public static final and the methods in an interface are by default public.
  - The default modifier is accessible only within package.

Access Modifiers in java

	public	private	protected	< unspecified >
class	allowed	not allowed	not allowed	allowed
constructor	allowed	allowed	allowed	allowed
variable	allowed	allowed	allowed	allowed
method	allowed	allowed	allowed	allowed

	class	subclass	package	outside
private	allowed	not allowed	not allowed	not allowed
protected	allowed	allowed	allowed	not allowed
public	allowed	allowed	allowed	allowed
< unspecified >	allowed	not allowed	allowed	not allowed

Reference

Notes by Adil Aslam