Kotlin

Mme Mounira Zouaghi

Cours V3.0

1. kotlin les notions OO

Plan de cours

1. **Les classes java vs les classes kotlin**
2. **La visibilité en kotlin**
3. **Les getters et les setters**
4. **Personnalisation d´une classe kotlin**
5. **L´heritage en kotlin**

4. Les classes

Des Classes java vers les classes kotlin

```java
public class user {

    // PROPERTIES
    private String email;
    private String password;
    private int age;

    // CONSTRUCTOR
    public user(String email, String password, int age){
        this.email = email;
        this.password = password;
        this.age = age;
    }

    // GETTERS
    public String getEmail() { return email; }
    public String getPassword() { return password; }
    public int getAge() { return age; }

    // SETTERS
    public void setEmail(String email) { this.email = email; }
    public void setPassword(String password) { this.password = password; }
    public void setAge(int age) { this.age = age; }

}
```

Des Classes java vers les classes kotlin

la visibilite par defaut est public

```
class userKt (var email:String,var password:String,var age:Int)
```

Instanciation

Pas de new

```
val myuser = userKt( email: "test",  password: "test",  age: 23)

println("MyUser:${myuser.email} ${myuser.password} ${myuser.age}")

myuser.age=25
```

```
MyUser:test test 23
```

Getter existe par defaut

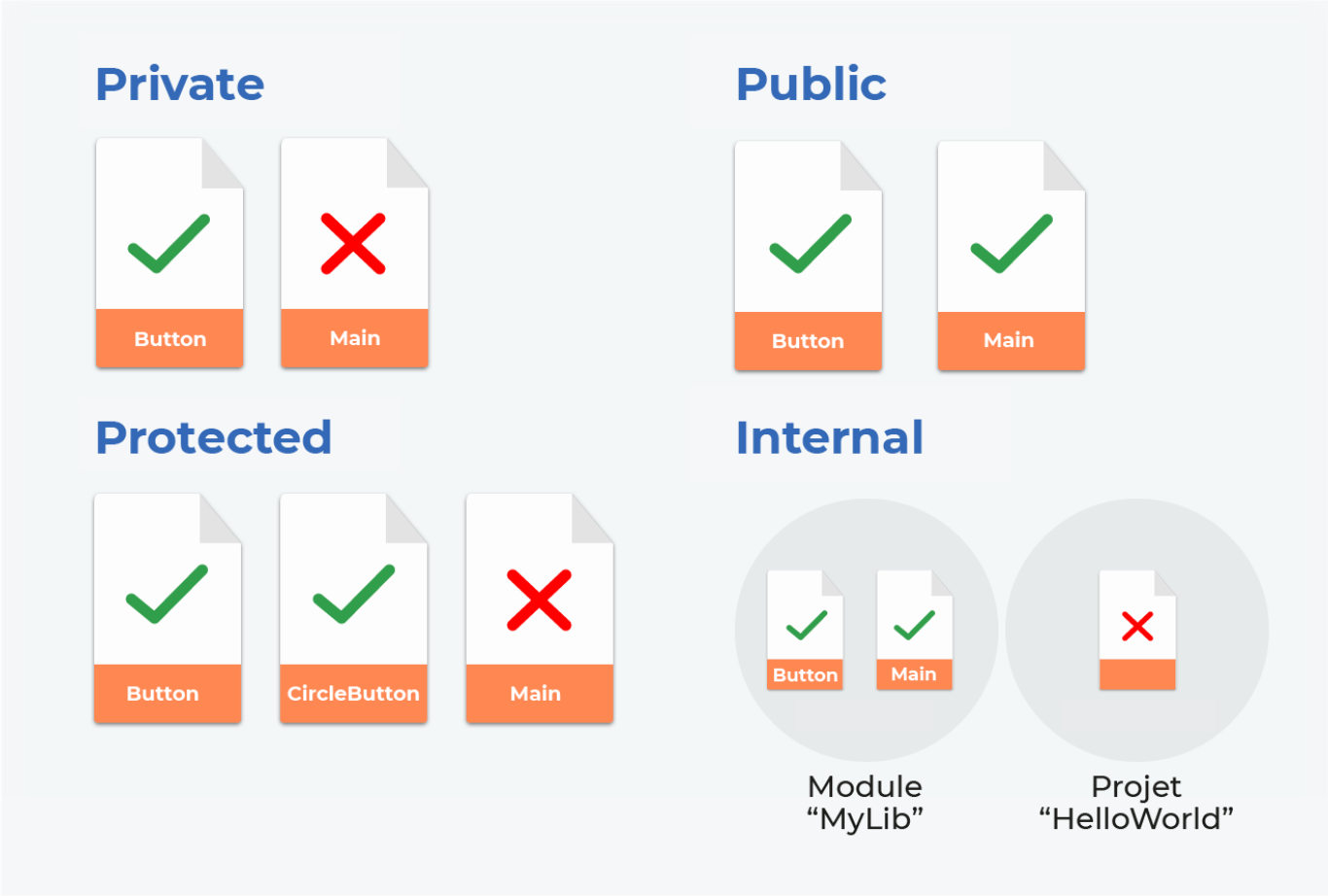
Setter existe par defaut

Classe immuable . Les proprietes sont modifiables

Visibilité

- **la visibilité par défaut** de n'importe quel élément de votre code (variables, fonctions, classes, etc.) est `public`
- on retrouve en Kotlin 4 principaux **modificateurs de visibilité** pour les membres (variables, fonctions, etc.) d'une classe :
- `private` : Un membre déclaré comme `private` sera visible uniquement dans la classe où il est déclaré.
- `protected` : Un membre déclaré comme `protected` sera visible uniquement dans la classe où il est déclaré ET dans ses sous-classes (via l'héritage).
- `internal` : Un membre déclaré comme `internal` sera visible par tous ceux du même module. Un module est un ensemble de fichiers compilés ensemble (comme une librairie Gradle ou Maven, par exemple).
- `public` : Un membre déclaré comme `public` sera visible partout et par tout le monde.

Visibilité


Private
Button
Main
Public
Button
Main
Protected
Button
CircleButton
Main
Internal
Button
Main
Module
“MyLib”
Projet
“HelloWorld”

Getter et setters

- **pas besoin de les déclarer explicitement**. Ils seront "générés" automatiquement grâce aux mots-clés `val` et `var` indiqués avant chaque nom de propriété dans le constructeur de la classe :
- `val` : La propriété sera **immuable**, vous ne pourrez donc pas la modifier. Kotlin générera alors pour vous uniquement un *getter*.
- `var` : La propriété sera **muable**, vous pourrez la modifier. Kotlin générera alors pour vous un *getter* et un *setter*.

Cas Pratique

Personnaliser une classe Kotlin

```kotlin
// declarer email comme parametre de constructeur
class User(email: String, var password: String, var age: Int){
    var email: String = email // declarer l'attribut field et l'initialiser
        // faire le getter personnalise
        get() {
            println("User is getting his email.");
            return field
        }
        // faire le setter personnalise
        set(value) {
            println("User is setting his email");
            field = value
        }
}
```

Ajouter des constructeurs

- Une classe kotlin peut avoir un ou plusieurs constructeurs

- Le constructeur primaire et des constructeurs secondaires qui invoquent explicitement le constructeur primaire

Ajouter des constructeurs

```
class userKt (var email: String, var password: String, var age: Int)//constructor par defaut
{
    var email2: String=""

    constructor (email: String) : this(email, password: "", age: 0)//constructor secondaire
    //constructor secondaire
    constructor(email: String, password: String, age: Int,email2:String ) : this(email,password,age)
    {
        this.email2=email2
    }

}
```

Pas de var ou val pour le constructeur secondaire

Cas Pratique

Les classes: héritage

Par defaut une classe kotlin est final

```
open class Personne (var name:String ,var surname:String )

class Etudiant(name:String,surname:String,var Matricule:String):Personne(name,surname)
```

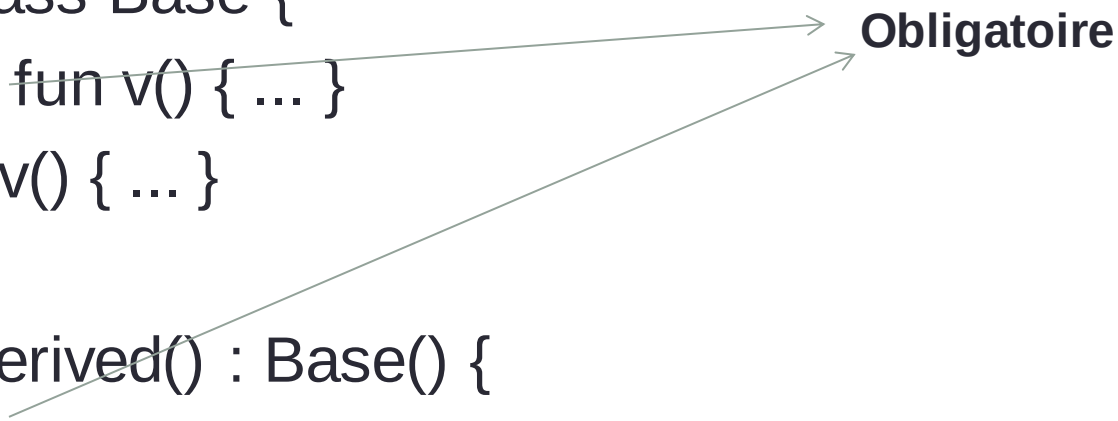
Ajouter le mot cle open

```
open class Personne (var name:String ,var surname:String )

//without constructeur primaire
class Etudiant:Personne{
    constructor(name:String ,surname:String ,Matricule:String) : super(name,surname)
    var Matricule:String
        get() {
            return  Matricule
        }
       set(value)
       {
           Matricule=value
       }
}
```

Override Methods

```
open class Base {
    open fun v() { ... }
    fun nv() { ... }
}
class Derived() : Base() {
    override fun v() { ... }
}
```



Override Methods

```
open class Personne (var name:String ,var surname:String )
{

    open fun afficheIdenty(){
        println("Identite: ${name.uppercase()} ${surname.uppercase()}")
    }
}



class Etudiant:Personne{

    override fun afficheIdenty() {
        super.afficheIdenty()
        println("Matricule:${Matricule}")
    }
```

Cas Pratique