

86

Programmation orienté Objet pour Swift

- La programmation orientée objet est un paradigme de programmation qui organise le code autour des "objets".
- Un objet est une instance d'une classe, qui peut contenir des données (propriétés) et des comportements (méthodes).

17/02/2025

86

Classes où structures



87

□ Exemple structure:

Créer un struct Personne avec les propriétés : nom, age, sexe, niveauScolaire

Créer quelques personnes les mettre dans un tableau


Ecrire une fonction qui prend en paramètre un tableau des personnes et qui renvoie un tableau composé des personnes de même niveau

17/02/2025

abdellatif.istef@estun5.ac.ma

87

Plan



88


1. Encapsulation
2. Abstraction
3. Héritage
4. Polymorphisme

17/02/2025

abdelilah.istef@estum5.ac.ma

88

Déclaration d'une classe



89

```
■ Class exemple{  
  //attributs  
  
  Func  
}
```

17/02/2025

abdelilah.istef@estum5.ac.ma

89

? Classes et Objets en Swift

90

```
class Personne {
    // Propriétés
    var nom: String
    var age: Int
    // Initialiseur
    init(nom: String, age: Int) {
        self.nom = nom
        self.age = age
    }
    // Méthode
    func sePresenter() {
        print("Bonjour, je m'appelle \(nom) et j'ai \(age) ans.")
    }
}
```

17/02/2025

abdelilah.iafer@estum5.ac.ma

90

Création d'un Objet

91

- Pour créer un objet (instance d'une classe), vous utilisez l'initialiseur.

```
//Création d'un Objet
let personne1 = Personne(nom: "Alami", age: 22)
personne1.sePresenter() // Output: Bonjour, je m'appelle
Alice et j'ai 30 ans.
```

17/02/2025

abdelilah.iafer@estum5.ac.ma

91

Encapsulation

92

- L'encapsulation permet de cacher les détails internes d'un objet.
- En Swift, vous pouvez utiliser des modificateurs d'accès comme **private**, **fileprivate**, **internal**, **public**, et **open** pour contrôler l'accès aux propriétés et méthodes.

17/02/2025

abdellil.lafar@estum5.ac.ma

92

Encapsulation

93

- Le Swift permet l'encapsulation par des marqueurs de visibilité (public, private et internal par défaut).
 - **private** : visibilité de l'entité seulement depuis le fichier
 - **internal** : visibilité de l'entité depuis d'autres fichiers sources du projet
 - **public** : visibilité de l'entité depuis d'autres fichiers sources du projet et depuis les imports (plus haut niveau)

17/02/2025

abdellil.lafar@estum5.ac.ma

93

Encapsulation

```
//Encapsulation
class CompteBancaire {
private var solde: Double = 0.0

func deposer(montant: Double)
{solde += montant}

func retirer(montant: Double) -
> Bool {
if montant <= solde {
solde -= montant
return true
} else {
return false
}
}
func getSolde() -> Double {
return solde
}}
```

```
let compte = CompteBancaire()
compte.deposer(montant: 1000)
print(compte.getSolde())
// Output: 1000.0
```

17/02/2025

abdelilah.iafer@estum5.ac.ma

94

Héritage

95

- L'héritage permet à une classe d'hériter des propriétés et méthodes d'une autre classe.
- En Swift, une classe peut hériter d'une seule classe (héritage simple).

17/02/2025

abdelilah.iafer@estum5.ac.ma

95

Héritage

96

```
Class class_fille : classe_mere{
    //attributs et méthodes héritées
    //propre attributs et méthodes
}
```

L héritage multiple est interdit en Swift

Si la classe mère a un constructeur la class fille le possèdera après l'héritage

17/02/2025

abdellilah.iafer@est.univ5.ac.ma

96

Héritage

97

```
//Héritage
class Animal {
    var nom: String

    init(nom: String) {self.nom = nom}
    func faireDuBruit() {
        print("\(nom) fait un bruit.")
    }

    class Chien: Animal {
        override func faireDuBruit() {
            print("\(nom) aboie.")
        }
    }

    let chien = Chien(nom: "Rex")
    chien.faireDuBruit()
    // Output: Rex aboie.
```

17/02/2025

abdellilah.iafer@est.univ5.ac.ma

97

Polymorphisme

98

- Le polymorphisme permet à des objets de différentes classes d'être traités comme des objets d'une classe commune.
- Cela est souvent réalisé grâce à l'héritage et à la redéfinition de méthodes.

17/02/2025

abdellil.lafar@estum5.ac.ma

98

Polymorphisme

99

```
//Polymorphisme
class Chat: Animal {
  override func faireDuBruit() {
    print("\(nom) miaule.")
  }
}

let animaux: [Animal] = [Chien(nom: "Rex"),
  Chat(nom: "Mimi")]

for animal in animaux {
  animal.faireDuBruit()
}

// Output:
// Rex aboie.
// Mimi miaule.
```

17/02/2025

abdellil.lafar@estum5.ac.ma

99

Polymorphisme

100

- Les méthodes publiques seront héritées et peuvent être réimplémenté par la notation « **override** » (overriding – polymorphisme en anglais).

```
Override func nom_fonct{

    //nouvelles instructions

}
```

17/02/2025

abdellil.lafor@estum5.ac.ma

100

Polymorphisme

101

- Si on conserve le corps de la méthode de base et on ajoute d'autres instructions:

```
Override func nom_fonct{

    super.Nom_methode
    //nouvelles instructions

}
```

17/02/2025

abdellil.lafor@estum5.ac.ma

101

Les protocoles

102

- Les protocoles (\Leftrightarrow interface en Java) est un modèle que va suivre une classe.
- Certain objets peuvent avoir des fonctionnalités communes
- Il est possible de spécifier des méthodes ainsi que des champs. L'ensemble du corps du protocole devra être réimplémenté par la nouvelle classe.

17/02/2025

abdellil.lafor@estum5.ac.ma

102

Les protocoles

103

- Exemple: les objets carré, rectangle et triangle.
- Ces objets peuvent avoir des fonctionnalités commune : calculer Air et périmètre

17/02/2025

abdellil.lafor@estum5.ac.ma

103

Les protocoles

104

- Déclarer un protocole:
 - ▣ Ecrire uniquement l'entête des fonctions

```
func NomMethode(arg1:type,arg2:type ...)
```

Le corps des méthodes sera définie dans les classes qui seront associé a ce protocole

17/02/2025

abdellil.lafor@estum5.ac.ma

104

Les protocoles

105

- Créer un protocole en swift:

```
protocol NomProtocole{
```

```
    // entête des methodes qui seront définies dans les classe  
    associé au protocole
```

```
}
```

17/02/2025

abdellil.lafor@estum5.ac.ma

105

Les protocoles

106

- Utiliser un protocole:

```
class NomClass:NomProtocole{
    // définition des méthodes dans l'entête est présent dans
    le protocole
    // les méthodes propre à la classe

}
```

17/02/2025

abdellil.lafor@estum5.ac.ma

106

Les protocoles

107

- Associé une classe à plusieurs protocoles:

```
class NomClass:NomProtocole1, NomProtocole2, ... {
    // définition des méthodes dans l'entête est présent dans
    le protocole
    // les méthodes propre à la classe

}
```

17/02/2025

abdellil.lafor@estum5.ac.ma

107

Les protocoles

108

- Associé une classe à plusieurs protocoles et a une classe mère:

```
class NomClass:NomClasseMere,NomProtocole1,
NomProtocole2, ... {
    // définition des méthodes dont l'entête est présent dans
    le protocole
    // les méthodes propre à la classe

}
```

17/02/2025

abdellil.iafar@estum5.ac.ma

108

109

- Comparer deux objets s'ils sont de même instance
(`===`) où (`!==`)

17/02/2025

abdellil.iafar@estum5.ac.ma

109

Les variables optionnelle

110

- Une optionnelle est une variable qui peut être affectée ou non à une valeur. Une variable optionnelle se voit attribuée une valeur « **nil** » par défaut grâce au « **?** ». Au contraire si notre optionnelle doit avoir une valeur sûr on lui attribue « **!** »

17/02/2025

abdellil.lafar@estum5.ac.ma

110

111

- `var jeSuisOptionnel: Int?` // Ceci est un optionnel
- `jeSuisOptionnel` // Le Playground affiche nil
- `var jeNeSuisPasOptionnel: Int` // Ceci est un entier
- `jeNeSuisPasOptionnel` // Le Playground plante...

17/02/2025

abdellil.lafar@estum5.ac.ma

111

Les optionnelles

112

- Les variables de type optionnel ont:
 - ▣ Soit une valeur
 - ▣ Soit pas de valeur -> nil
 - ▣ Voir exemple

17/02/2025

abdellil.lafar@estum5.ac.ma

112

Déclarer une variable Les optionnelle

113

- Var opt: Int? = 7
 - ▣ Opt = nil
 - ▣ Opt = 22
 - Une chose qui est impossible pour les autres variables, pas de changement de type de valeur

17/02/2025

abdellil.lafar@estum5.ac.ma

113