

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

جامعة شعيب الدكالي
الكلية المتعددة التخصصات بميدون
F.P.S.B
Université Chouaib Doukkali
Faculté Polydisciplinaire - Sidi Bennour

Conception orientée objet



B. Jabir
ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

1

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Composante(s) du module	Volume horaire (VH)					
	Cours	TD	TP	Activités Pratiques (Travaux de terrain, Projets, Stages, ...), Autres/préciser)	Travail personnel	Evaluation des connaissances
ORIENTEE OBJET (UML)	12	10			1,5	23,5
Java	12		13		1,5	26,6
VH global du module	24	10	13		3	50
% VH	48%	20%	26%		6%	100%

2

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Plan:

- ❑ Introduction à la conception orientée objets.
- ❑ Les éléments de l'approche objet
- ❑ Diagramme des cas d'utilisation.
- ❑ Diagramme de classes.
- ❑ Diagramme d'objets.
- ❑ Diagramme de séquences.
- ❑ Diagramme des composants.
- ❑ Diagramme des packages.
- ❑ Diagramme d'état transition.
- ❑ Diagramme d'activités.

3

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Introduction:

La qualité d'un logiciel se mesure par rapport à plusieurs critères :

- Répondre aux spécifications fonctionnelles :
 - Une application est créée pour répondre , tout d'abord, aux besoins fonctionnels des entreprises.
- Les performances:
 - La rapidité d'exécution et Le temps de réponse,
 - Doit être bâtie sur une architecture robuste
 - Éviter le problème de montée en charge
- La maintenance:
 - Une application doit évoluer dans le temps.
 - Doit être fermée à la modification et ouverte à l'extension
 - Une application mal conçue est difficile à maintenir, par suite elle finit un jour à la poubelle.

4

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Introduction:

La qualité d'un logiciel se mesure par rapport à plusieurs critères : (suite)

- Sécurité
 - ▶ Garantir l'intégrité et la sécurité des données
- Portabilité
 - ▶ Doit être capable de s'exécuter dans différentes plateformes.
- Capacité de communiquer avec d'autres applications distantes.
- Disponibilité et tolérance aux pannes
- Capacité de fournir le service aux différents types de clients :
 - ▶ Client lourd : Interfaces graphiques SWING
 - ▶ Interface Web : protocole Http
 - ▶ Téléphone : SMS .
 - ▶ ...
- Design de ses interfaces graphiques
 - ▶ Charte graphique et charte de navigation
 - ▶ Accès via différentes interfaces (Web, Téléphone, PDA, ,)

Continuation

5

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Introduction:

Pour apporter une réponse à tous ces problèmes, le génie logiciel s'intéresse particulièrement à la manière dont le code source d'un logiciel est spécifié puis produit. Ainsi, le génie logiciel touche au cycle de vie des logiciels :

- ▶ l'analyse du besoin,
- ▶ l'élaboration des spécifications,
- ▶ Modélisation et conception,
- ▶ le développement,
- ▶ la phase de test,
- ▶ la maintenance.

6

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

La modélisation vs La conception

- La **modélisation** consiste à représenter un système ou un processus de manière abstraite à l'aide de schémas et de diagrammes. Elle permet de visualiser et de comprendre les différentes composantes et interactions du système. Dans le domaine des logiciels, la modélisation peut être utilisée pour représenter l'architecture logicielle, les flux de données, les cas d'utilisation, etc.
- La **conception**, quant à elle, consiste à transformer les modèles abstraits en une représentation concrète du système. Elle se concentre sur les détails de l'implémentation, tels que l'organisation du code, la définition des classes, des méthodes, des attributs, etc. La conception permet de décrire comment le système sera réellement mis en œuvre.

• Modélisation (Vision métier)

- Représente les besoins fonctionnels
- Focus sur le "QUOI"
- Plus abstraite

• Conception (Vision technique)

- Détaille l'implémentation technique
- Focus sur le "COMMENT"
- Plus concrète, proche du code

7

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Vues multiples (aspects d'un système logiciel)

```

graph TD
    House((House)) --- ViewPlombier[Vue du plombier]
    House --- ViewElectricien[Vue de l'électricien]
    House --- ViewLocataire[Vue du locataire]
    House --- ViewProprietaire[Vue du propriétaire]
    House --- ViewArchitecte[Vue de l'architecte]
    House --- ViewService[Vue du service des impôts locaux]
    House --- ViewCadastral[Vue du cadastre]
    House --- ViewNotaire1[Vue du notaire]
    House --- ViewMacon[Vue du maçon]
    House --- ViewNotaire2[Vue du notaire]
  
```

8

Brahim Jaber ibra.jaber@gmail.com/jaber.ibrahim@ucd.ac.ma

Présentation d'UML

- UML est l'*Unified Modeling Language* standardisé par l'**OMG** (*Object Management Group* : <http://www.omg.org>). Ce n'est pas une méthode, il ne donne pas de solution pour la mise en œuvre d'un projet. C'est avant tout un **formalisme graphique** issu de notations employées dans différentes méthodes objets.
- UML sert à :
 - Décomposer le processus de développement,
 - Mettre en relation les experts métiers et les analystes,
 - Cordonner les équipes d'analyse et de conception,
 - Séparer l'analyse de la réalisation,
 - Prendre en compte l'**évolution** de l'analyse et du développement,
 - Migrer facilement vers une architecture objet d'un point de vue statique et dynamique.

9

Brahim Jaber ibra.jaber@gmail.com/jaber.ibrahim@ucd.ac.ma

Présentation d'UML

- Décomposer le processus de développement :**
 - Lors du développement d'un nouveau logiciel, une équipe utilise UML pour décomposer le processus en étapes plus petites et gérables. Par exemple, ils peuvent commencer par la création d'un diagramme de cas d'utilisation pour définir les fonctionnalités attendues.
- Mettre en relation les experts métiers et les analystes :**
 - Les experts métiers, qui ont une connaissance approfondie du domaine pour lequel le logiciel est développé, collaborent avec les analystes en utilisant UML pour définir les besoins et les spécifications. Par exemple, un expert métier peut expliquer les exigences à l'analyste, qui les traduira ensuite en diagrammes UML.
- Coordonner les équipes d'analyse et de conception :**
 - Dans un projet, l'équipe d'analyse utilise UML pour créer des modèles conceptuels, tandis que l'équipe de conception utilise UML pour élaborer des modèles de conception détaillés. Les diagrammes UML servent de langage commun pour coordonner ces deux équipes. Par exemple, l'équipe d'analyse peut fournir un diagramme de classe UML à l'équipe de conception pour guider l'implémentation.

10

Brahim Jaber ibra.jaber@gmail.com/jaber.ibrahim@ucd.ac.ma

Présentation d'UML

- Séparer l'analyse de la réalisation :**
 - UML permet de séparer l'analyse des besoins du logiciel de la phase de réalisation. Par exemple, lors de la modélisation, l'équipe peut se concentrer sur les diagrammes de classes, d'activités et de séquence pour décrire le système sans se préoccuper des détails d'implémentation.
- Prendre en compte l'**évolution** de l'analyse et du développement :**
 - UML facilite la prise en compte des changements tout au long du projet. Si de nouveaux besoins ou des modifications surviennent, les diagrammes UML peuvent être mis à jour pour refléter ces changements, et cela peut être fait de manière cohérente et compréhensible.
- Migrer facilement vers une architecture objet d'un point de vue statique et dynamique :**
 - UML offre des diagrammes statiques (comme les diagrammes de classes) et des diagrammes dynamiques (comme les diagrammes de séquence) qui aident à concevoir une architecture orientée objet. Par exemple, en utilisant UML, une équipe peut représenter les classes et les interactions entre les objets dans le système.
 - UML est utilisé pour créer des modèles abstraits de systèmes ou de processus afin de les comprendre plus facilement, de les documenter et de les communiquer efficacement

11

Brahim Jaber ibra.jaber@gmail.com/jaber.ibrahim@ucd.ac.ma

Historique : Évolution de UML

UML est le fruit de l'unification de 3 méthodes de modélisation orientées objet

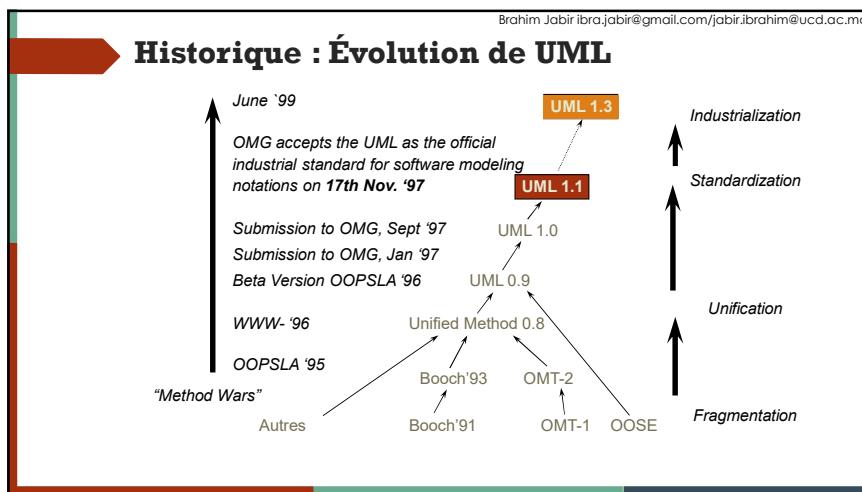
- OMT (Object Modeling technique) James Rumbaugh (1991)
- Booch: Grady Booch (1991)
- OOSE (Object Oriented Software Engineering): Ivar Jacobson (1992)

```

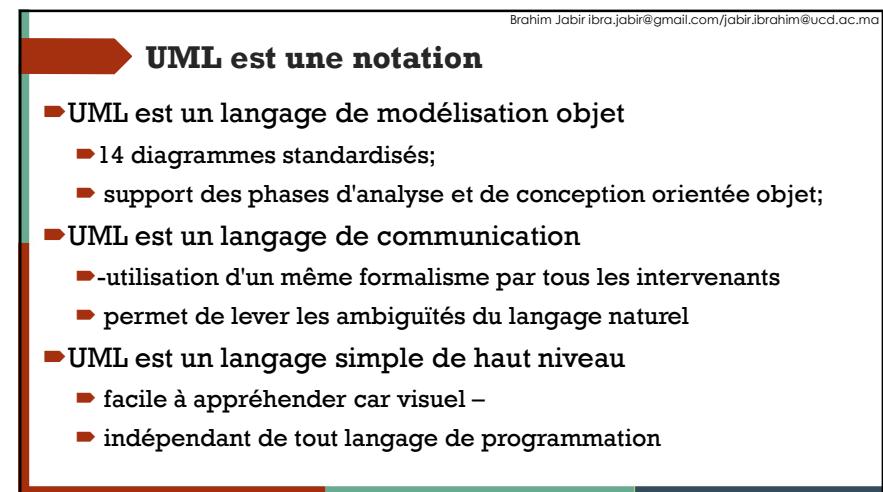
    graph TD
        OMT[OMT] --> UML[UML]
        Booch[Booch] --> UML
        OOSE[OOSE] --> UML
        UML["UML 1.0 a été normalisé en janvier 1997"]
    
```

Comprendre la logique d'évolution d'UML.

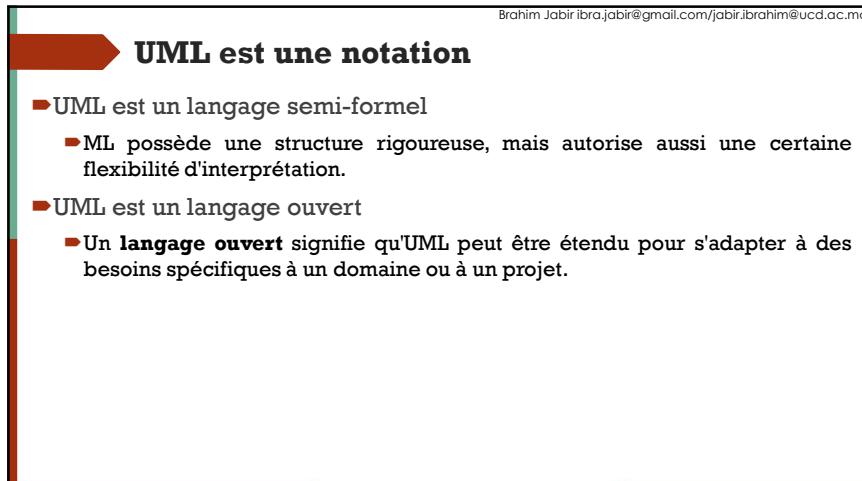
12



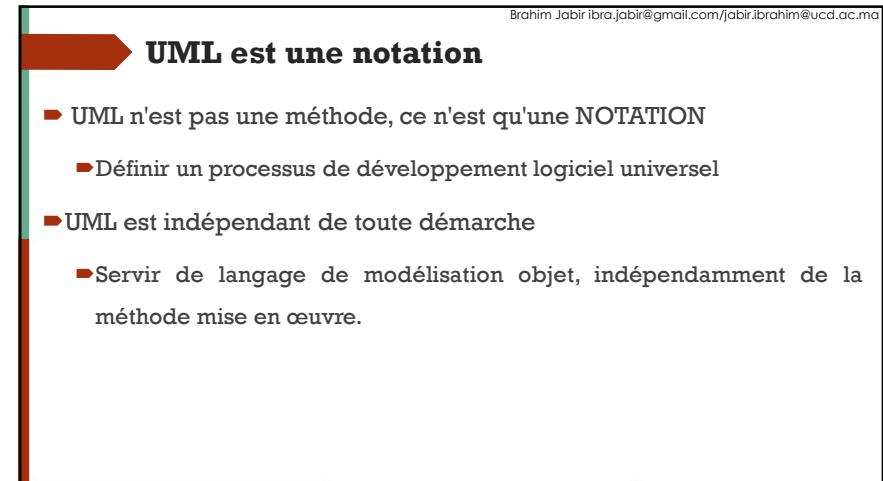
13



14



15



16

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Niveaux de modélisation

- Niveau << Système >>
 - ▶ recadrage contextuel du système dans son environnement
 - ▶ détermination des interactions avec les utilisateurs
 - ▶ modélisation de l'architecture des processus modélisation de l'architecture d'implémentation
- Niveau << Sous Système >>
 - ▶ décomposition structurelle hiérarchique du système
- Niveau << Entité>>
 - ▶ modélisation détaillée au niveau de l'objet

17

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Axes de modélisation

- ▶ Axe structurel
 - ▶ modélisation statique du système
 - ▶ quels objets manipule le système ?
 - ▶ détermination du **QUOI**
- ▶ Axe fonctionnel modélisation des traitements offerts par le système que fait le système ?
 - ▶ détermination du **COMMENT**
- ▶ Axe comportemental / dynamique
 - ▶ modélisation dynamique du système sous quelles conditions agit le système ?
 - ▶ détermination du **QUAND**

18

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Comment "réddiger" un modèle avec UML ?

- UML permet de définir et de visualiser un modèle, à l'aide de diagrammes.
- Par extension et abus de langage, « un diagramme UML est aussi un modèle » mais un diagramme modélise un aspect du modèle global.
- Chaque type de diagramme UML possède une structure (les types des éléments de modélisation qui le composent sont prédéfinis).
- Un type de diagramme UML véhicule une sémantique précise (un type de diagramme offre toujours la même vue d'un système).
- Les différents types de diagrammes UML offrent une vue complète des aspects statiques et dynamiques d'un système.

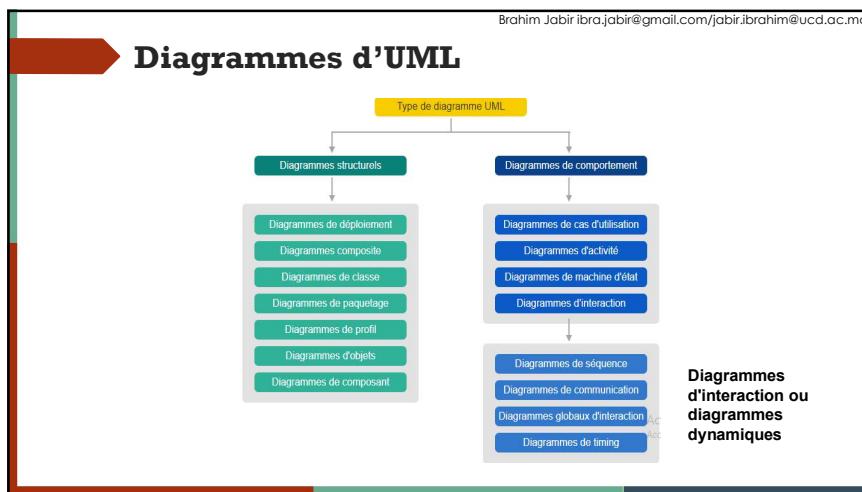
19

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

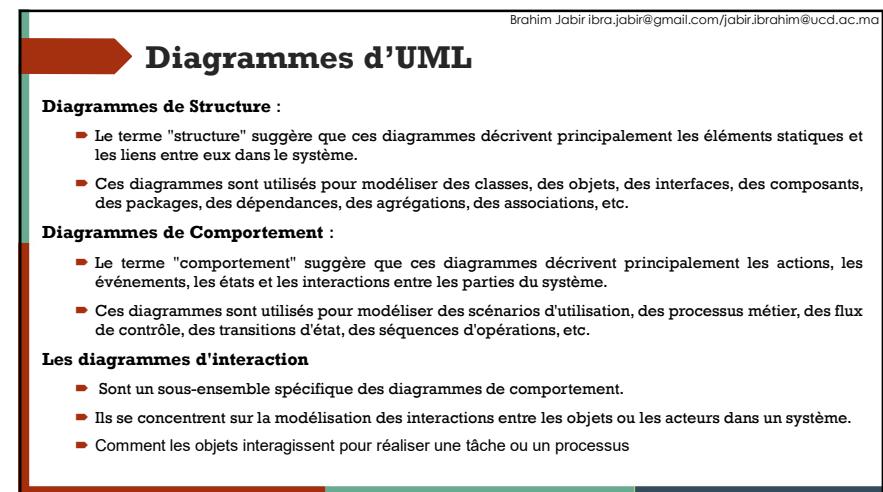
Comment "réddiger" un modèle avec UML ?

- **Modèle UML** : Il représente l'ensemble des informations et de la structure du système logiciel, généralement composé de plusieurs diagrammes UML. Le modèle UML est une abstraction conceptuelle qui décrit le système dans son ensemble.
- **Diagramme UML** : Il s'agit d'une représentation graphique spécifique d'une partie du modèle UML. Chaque type de diagramme UML (comme un diagramme de classe, un diagramme de séquence, etc.) est conçu pour modéliser un aspect précis du système. (utilisation des diagrammes pour modéliser le modele)

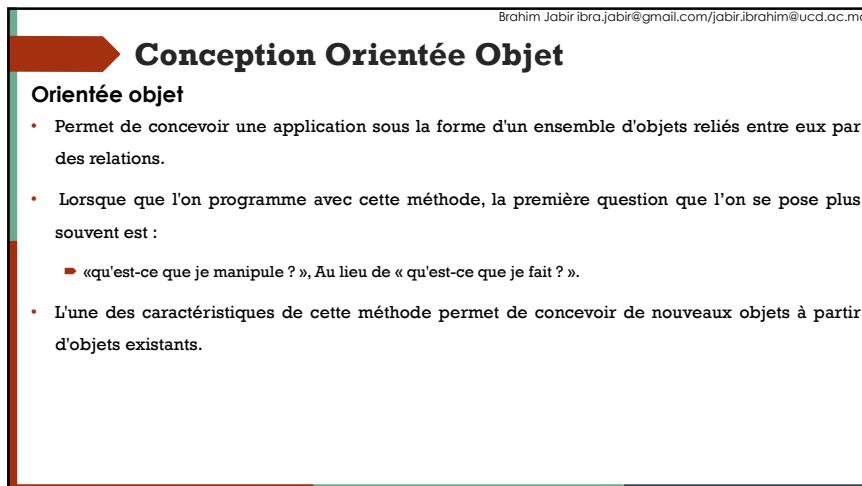
20



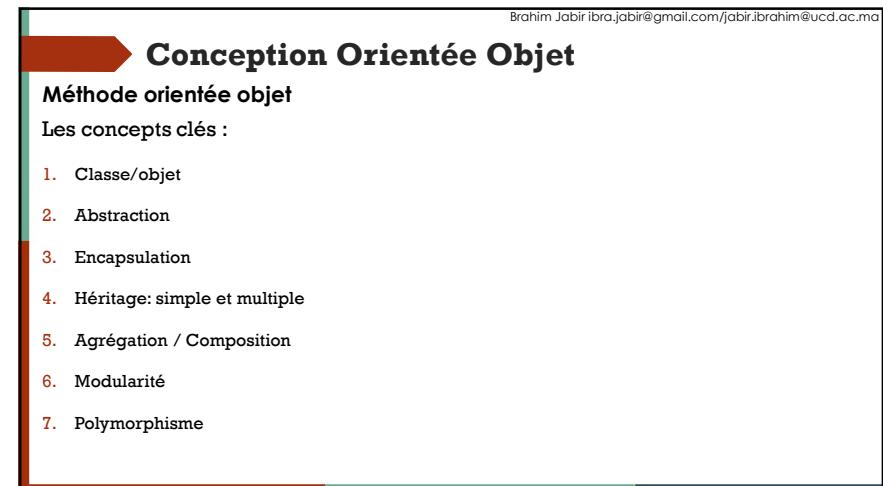
21



22



23



24

Les éléments de l'approche objet

Classe/objet

- **Une classe** est un modèle ou un plan qui définit les attributs (variables) et les méthodes (fonctions) qui seront partagés par les objets créés à partir de cette classe.
- Elle sert de modèle pour créer des objets.
 - Attributs : - données dont les valeurs représentent l'état de l'objet
 - Méthodes: - opérations applicables aux objets

25

Les éléments de l'approche objet

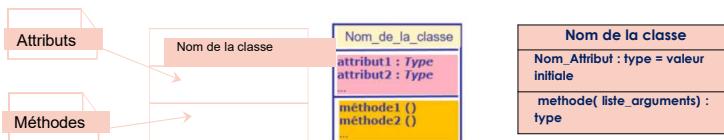
Classe

- 1, Les attributs permettent de décrire l'état des objets de cette classe.
 - Chaque attribut est défini par: (Nom_Attribut : type = valeur initiale)
 - Son nom
 - Son type
 - Éventuellement sa valeur initiale
- 2, Les méthodes permettent de décrire le comportement des objets de cette classe.
 - Une méthode représente une procédure ou une fonction qui permet d'exécuter un certain nombre d'instructions.
 - Parmi les méthodes spéciales d'une classe :
 - Une méthode qui est appelée au moment de la création d'un objet de cette classe. Cette méthode est appelée **CONSTRUCTEUR**

26

Les éléments de l'approche objet

Représentation d'une classe en UML



- Une classe est représentée dans UML par un rectangle contenant le nom de la classe.
- Les compartiments d'une classe peuvent être omis si leur contenu n'est pas pertinent dans le contexte d'un diagramme de classe.
- Les noms d'attributs d'une classe sont uniques.
- Une opération définit une fonction appliquée à des objets d'une classe.

27

Les éléments de l'approche objet

Représentation d'une classe en UML

Types de données primitifs :

- int : Entier.
- float : Nombre à virgule flottante.
- double : Nombre à virgule flottante double précision.
- boolean : Valeur booléenne (vrai ou faux).
- char : Caractère unique.

Types de données chaîne :

- String : Chaîne de caractères.

Types de données date/heure :

- Date : Date (jour, mois, année).
- Time : Heure du jour.
- DateTime : Date et heure combinées.

28

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Les éléments de l'approche objet

Donner des exemples d'attributs et d'opérations pour ces classes ?

- Classe Voiture :
- Classe Client :
- Classe Animal :
- Classe Compte Bancaire :
- Classe Produit :
- Classe Enseignant :

29

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Les éléments de l'approche objet

Classe Voiture :	Classe Personne	Classe Client :	Classe Animal :	Classe Compte Bancaire :	Classe Produit :	Classe Enseignant :
Attributs : Marque Modèle Année de fabrication Numéro de série Couleur Prix Opérations (méthodes) : Démarrer Arrêter Accélérer Freiner ObtenirInformation (pour récupérer des détails sur la voiture)	Attributs : Nom Prénom Date de naissance Adresse Numéro de téléphone Adresse e-mail Opérations (méthodes) : Se présenter Changer d'adresse Changer de numéro de téléphone Envoyer un e-mail	Attributs : ID client Nom de l'entreprise (le cas échéant) Liste de commandes (peut être une liste d'objets Commande)	Attributs : Nom Espèce Âge Poids Lieu de résidence (zoo, habitat naturel, etc.) Régime alimentaire Opérations (méthodes) : Passer une commande Consulter l'historique des commandes Mettre à jour	Attributs : Numéro de compte Solde Titulaire du compte (peut être une référence à la classe Personne) Opérations (méthodes) : Déposer de l'argent Manger Dormir Se déplacer Émettre un son	Attributs : Code produit Nom du produit Description Prix Quantité en stock Opérations (méthodes) : Mettre à jour les détails du produit Ajouter au stock Retirer du stock Calculer le montant total (prix x quantité)	Attributs : Nom Prénom Matière enseignée Années d'expérience Liste d'étudiants enseignés (peut être une liste d'objets Étudiant) Opérations (méthodes) : Donner un cours Évaluer les étudiants Publier des notes

30

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Les éléments de l'approche objet

Exemples de classe:

1- Une personne est définie par:

- Son nom, son adresse et sa date de naissance.
- Les opérations que l'on peut appliquer à un objet Personne sont:
 - age() : méthode qui retourne l'âge de la personne
 - changerAdresse() : méthode qui permet de modifier l'adresse de la personne

2- Un point est défini par:

- Ses coordonnées x et y
- Les opérations l'on peut exécuter sur un objet Point sont:
 - toString() : opération qui retourne une chaîne de caractères de type Point(x,y)
 - distance(Point p) : opération qui retourne la distance entre le point et un autre point p.

3- Un segment est défini par :

- Deux points p1 et p2
- Les opérations de la classe Segment sont:
 - getLongueur() : opération qui retourne la longueur du segment
 - appartient(Point p) : retourne si un point p appartient au segment ou non

31

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Les éléments de l'approche objet

Objet :

- Un objet est une structure informatique définie par un état et un comportement
- **Objet=état + comportement + Identité**
- L'état regroupe les valeurs instantanées de tous les attributs de l'objet.
- Le comportement regroupe toutes les compétences et décrit les actions et les réactions de l'objet. Autrement dit le comportement est défini par les opérations que l'objet peut effectuer.
 - **L'état d'un objet peut changer dans le temps.**
 - **Le comportement qui modifie l'état de l'objet**

32

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Les éléments de l'approche objet

Objet :

- Un objet est une instance d'une classe, créée à partir du modèle défini par la classe.
- Réutilisation du Voiture :** La POO permet de réutiliser le modèle de classe pour créer de multiples objets similaires avec des données et des comportements spécifiques. Cela favorise la modularité et la maintenabilité du code.
- Entité autonome :** Chaque objet est autonome et isolé des autres objets de la même classe. Les modifications apportées à un objet n'affectent pas automatiquement les autres objets de la classe.

Notation:
→ un Objet est une instantiation (occurrence) d'une classe

```

graph LR
    subgraph Etat [Etat de l'objet]
        direction TB
        A[Couleur="rouge"]
        B[Carburant=20]
        C[Puissance=120]
        D[État de l'objet]
        E[Comportement]
    end
    subgraph Comportement [Comportement]
        direction TB
        F[demarrer()]
        G[accelerer()]
        H[freiner()]
    end
    A --- v1["v1:Voiture"]
    B --- v1
    C --- v1
    D --- v1
    E --- v1
    F --- v1
    G --- v1
    H --- v1
  
```

33

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Les éléments de l'approche objet

Objet :

Un objet est une instance concrète d'une classe. Chaque objet appartient à une classe spécifique et possède ses propres valeurs d'attributs, ce qui le rend unique. Les objets représentent des entités spécifiques dans un programme.

Généralités et Particularités : Les classes contiennent généralement des généralités, c'est-à-dire les caractéristiques communes à tous les objets de cette classe, tandis que les particularités, telles que les valeurs spécifiques des attributs, sont détenues par chaque objet individuel de cette classe.

Instanciation : L'instanciation est le processus de création d'un objet à partir d'une classe. Lorsque vous instanciez une classe, vous créez une copie de cette classe avec ses attributs et méthodes, prête à être utilisée. Chaque objet est une instance d'une classe.

34

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Les éléments de l'approche objet

Identité d'un objet:

- En plus de son état, un objet possède une identité qui caractérise son existence propre.
- Cette identité s'appelle également référence ou handle de l'objet
- En terme informatique de bas niveau, l'identité d'un objet représente son adresse mémoire.
- Deux objets ne peuvent pas avoir la même identité: c'est-à-dire que deux objets ne peuvent pas avoir le même emplacement mémoire.

35

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Les éléments de l'approche objet

Identité d'un objet:

```

classDiagram
    class Personne {
        nom : String
        adresse : String
        dateNaissance:Date
        age() : Int
        changerAdresse(String a) : void
    }
    Personne1 : Personne
    Personne2 : Personne
    Personne1 --> Personne
    Personne2 --> Personne
  
```

- "personne1" et "personne2" sont des instances de la classe "Personne" avec des informations spécifiques sur deux personnes différentes, y compris leur nom, adresse et date de naissance.

36

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Les éléments de l'approche objet

Classe/Objet | Résumé

- Les objets qui ont des caractéristiques communes sont regroupés dans une entité appelé **classe**.
- La classe décrit le domaine de définition d'un ensemble d'**objets**.
- Chaque objet appartient à une classe.
- Les généralités sont contenues dans les classes et les particularités dans les objets.
- Les objets sont construits à partir de leur classe par un processus qui s'appelle l'**instanciation**.
- Tout objet est une instance d'une classe.

37

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Les éléments de l'approche objet

Étude de cas : Gestionnaire de bibliothèque

Description : Vous êtes confrontés à la tâche de créer un système de gestion pour une bibliothèque. L'objectif principal de ce système est de permettre aux bibliothécaires de gérer efficacement les livres, les emprunts et les membres de la bibliothèque. Pour ce faire, vous devez concevoir les classes nécessaires pour représenter ces entités, définir les attributs pertinents pour chaque classe et spécifier les opérations qui permettront aux utilisateurs du système d'interagir avec ces entités.

38

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Les éléments de l'approche objet

Héritage

L'héritage permet de créer de nouvelles classes en se basant sur des classes existantes, en héritant de leurs attributs et comportements.

- **Classe de base (superclasse) :** Commencez par créer une classe de base ou superclasse qui contient les attributs et méthodes communs que vous souhaitez partager avec les classes dérivées. Dans le diagramme de classes, cela se représente avec une boîte contenant le nom de la classe en haut et ses attributs et méthodes en dessous.
- **Classe dérivée (sous-classe) :** Ensuite, créez une classe dérivée ou sous-classe qui héritera de la classe de base. Dans le diagramme de classes, dessinez une flèche avec une ligne solide de la sous-classe vers la superclasse. La flèche indique la direction de l'héritage.

39

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Les éléments de l'approche objet

Héritage

- L'**héritage** est noté avec une flèche (quelques fois en pointillées) terminée par un triangle vide.
- La classe mère peut être une classe conventionnelle, une classe abstraite, ou encore une interface.
- On précise dans ce cas la nature de la classe mère afin d'éviter toute confusion.
- Le cas particulier d'héritage qu'est l'**implémentation** se représente avec des flèches en pointillés.

40

Brahim Jabir ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Les éléments de l'approche objet

Héritage

```

classDiagram
    class Animal
    class Vertebrate
    class Invertebrate
    class Ovide
    class Equide
    class Canide
    class Cheval
    class Girafe

    Animal <|-- Vertebrate
    Animal <|-- Invertebrate
    Vertebrate <|-- Ovide
    Vertebrate <|-- Equide
    Vertebrate <|-- Canide
    Equide <|-- Cheval
    Equide <|-- Girafe
  
```

Une flèche d'héritage en tirets indique qu'une classe raffine ou implémente une interface. (Ce concept sera abordé ultérieurement.)

Diagramme UML montrant l'héritage. La classe **Animal** est la superclass avec des subclasses **Vertebrate** et **Invertebrate**. **Vertebrate** a des subclasses **Ovide**, **Equide** et **Canide**, dont **Equide** a des subclasses **Cheval** et **Girafe**.

Diagramme UML montrant l'implémentation d'une interface. L'interface **Interface** définit une opération **+opere() : void**. La classe **Implementation** implémente cette interface et définit également l'opération **+opere() : void**.

41

Brahim Jabir ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Les éléments de l'approche objet

Héritage

UML

```

classDiagram
    class Article {
        +designation
        +prix
    }
    class Livre {
        +isbn
        +nbPages
    }
    class Disque {
        +label
    }
    class Video
    class Poche
    class Broche

    Article <|-- Livre
    Article <|-- Disque
    Article <|-- Video
    Livre <|-- Poche
    Disque <|-- Broche
    Article <|--+acheter()
  
```

Java

```

1 class Livre extends Article {
2 ...
3 }
  
```

- La classe enfant (celle qui hérite) possède toutes les propriétés de ses classes parents (attributs et opérations)
- La classe enfant est la classe spécialisée (ici Livre)
- La classe parent est la classe générale (ici Article)
- Elle n'a accès qu'aux propriétés publiques (comme tout le monde) et protégées de ses parents.

42

Brahim Jabir ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Les éléments de l'approche objet

Héritage

- Relation: spécialisation/généralisation.

Spécialisation : Ce terme est souvent utilisé pour mettre l'accent sur le fait que des classes filles sont créées à partir d'une classe parente pour représenter des cas spécifiques ou des sous-types de la classe parente. Par exemple, si vous avez une classe parente "Véhicule" et que vous créez des classes filles telles que "Voiture" et "Moto", vous effectuez une spécialisation pour représenter des types spécifiques de véhicules.

Généralisation : Ce terme est souvent utilisé pour mettre l'accent sur le fait que les classes filles héritent des caractéristiques générales de la classe parente. Dans ce contexte, la classe parente est considérée comme une généralisation des classes filles, car elle contient des attributs et des méthodes communs à toutes les classes filles.

43

Brahim Jabir ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Les éléments de l'approche objet

Héritage

Héritage Simple :

L'héritage simple, également connu sous le nom d'héritage à un seul parent, signifie qu'une classe enfant (ou sous-classe) peut hériter des attributs et des méthodes d'une seule classe parente (ou superset).

Dans un système d'héritage simple, une classe peut avoir une et une seule classe parente directe. Cela signifie que la hiérarchie des classes est en forme d'arborescence, avec une classe racine et des classes descendantes.

Héritage Multiple :

L'héritage multiple permet à une classe enfant de hériter des attributs et des méthodes de plusieurs classes parentes. Cela signifie qu'une classe peut avoir plusieurs classes parentes directes.

L'héritage multiple peut rendre la hiérarchie des classes plus complexe, car une classe enfant peut hériter de plusieurs sources différentes, chacune avec ses propres attributs et méthodes.

On prête aussi attention au fait que dans certains langages à objets, en particulier Java, l'héritage multiple est interdit.

44

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Les éléments de l'approche objet

Héritage

```

classDiagram
    class VehiculeMoteur {
        +demarrer()
        +arreter()
    }
    class VehiculeMarin {
        +naviguer()
    }
    class Personne {
        -nom
        -age
        #adresse
        +getAge()
    }
    class Etudiant {
        -numEtud
        +modifieAdr()
    }
    class Bateau {
        << VehiculeMoteur, VehiculeMarin >>
    }
    Bateau --> VehiculeMoteur
    Bateau --> VehiculeMarin
    Etudiant --> Personne
  
```

45

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Les éléments de l'approche objet

Héritage

```

// Classe parente (superclasse)
class Personne {
    String nom;
    // Constructeur de la classe parente
    public Personne(String nom) {
        this.nom = nom;
    }
    // Méthode de la classe parente pour se présenter
    public void sePresenter() {
        System.out.println("Je suis " + nom + ".");
    }
}

// Classe dérivée (sous-classe)
class Etudiant extends Personne {
    int numeroEtudiant;
    // Constructeur de la classe dérivée avec appel du constructeur de la classe parente
    public Etudiant(String nom, int numeroEtudiant) {
        super(nom); // Appel du constructeur de la classe parente
        this.numeroEtudiant = numeroEtudiant;
    }
    // Méthode spécifique de la classe dérivée pour étudier
    public void etudier() {
        System.out.println(nom + " étudie.");
    }
}

public class ExempleHeritage {
    public static void main(String[] args) {
        // Création d'un objet de la classe Etudiant avec nom "Alice" et numéro étudiant 12345
        Etudiant etudiant = new Etudiant("Alice", 12345);
        // Appel de la méthode de la classe parente pour se présenter
        etudiant.sePresenter();
        // Appel de la méthode spécifique de la classe dérivée pour étudier
        etudiant.etudier();
    }
}
  
```

46

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Les éléments de l'approche objet

Encapsulation

Les membres (attributs, opérations) d'une classe peuvent être protégés en utilisant quatre niveaux de protection :

- Public (+) : Accès à partir de toute entité interne ou externe à la classe
- Protégé (#) : Accès à partir de la classe ou des classes dérivées
- Privé (-) : accès à partir des opérations de la classe
- Autorisation par défaut : chaque langage de programmation définit une autorisation par défaut. Dans java, l'autorisation par défaut est package. Cela signifie que seules les classes du même package qui ont le droit d'accéder à ce membre.

47

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Les éléments de l'approche objet

Encapsulation

- - (moins) : Cela signifie que l'attribut est privé. Il est accessible uniquement à l'intérieur de la classe qui le contient. Aucune autre classe, même celles du même package, ne peut accéder directement à cet attribut.
- # (dièse) : Cela signifie que l'attribut est protégé. Il est accessible à l'intérieur de la classe qui le contient, ainsi qu'à ses sous-classes (classes qui héritent de cette classe) qu'elles soient dans le même package ou dans un package différent.
- + (plus) : Cela signifie que l'attribut est public. Il est accessible à partir de n'importe quelle classe, qu'elle soit dans le même package ou dans un package différent. En d'autres termes, il n'y a pas de restrictions d'accès à cet attribut.
- Aucun symbole : En Java, la visibilité par défaut (si aucun symbole n'est utilisé) est 'package-private,' mais en UML, on utilise généralement le symbole '-' devant les attributs ou méthodes pour représenter cette visibilité de package pour éviter toute confusion.

48

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Les éléments de l'approche objet

Encapsulation

- nom indique qu'il est privé et ne doit pas être directement accessible depuis l'extérieur de la classe où il est défini.
- +adresse : String : Cet attribut est défini avec une visibilité publique (+), ce qui signifie qu'il est accessible depuis n'importe quelle classe, y compris celles situées dans d'autres packages. Cela semble approprié si vous voulez que cet attribut soit largement accessible.
- #dateNaissance: Date : Cet attribut est défini avec une visibilité protégée (#), ce qui signifie qu'il est accessible uniquement depuis les classes dérivées (sous-classes) de cette classe. Cela peut être utile si vous souhaitez permettre aux sous-classes d'accéder à cet attribut, mais pas aux classes extérieures.
- +age() : int : Cette méthode est définie avec une visibilité publique (+), ce qui signifie qu'elle est accessible depuis n'importe quelle classe. Cela semble approprié si vous voulez que cette méthode soit utilisée largement.
- +changerAdresse(String a) : void : Cette méthode est également définie avec une visibilité publique (+), ce qui signifie qu'elle est accessible depuis n'importe quelle classe. Cela semble approprié si vous souhaitez permettre à d'autres classes de changer l'adresse de l'objet.

Personne
-nom: String
+adresse: String
#dateNaissance: Date
+age() : int
+changerAdresse(String a) : void

49

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Les éléments de l'approche objet

Encapsulation - Java

Attributs et Méthodes Publics :

```

1  public class Personne {
2      // Attribut public
3      public String nom;
4
5      // Constructeur
6      public Personne(String nom) {
7          this.nom = nom;
8      }
9
10     // Méthode publique pour afficher le nom
11     public void afficherNom() {
12         System.out.println("Nom : " + nom);
13     }
14 }
15
16 public class ProgrammePrincipal {
17     public static void main(String[] args) {
18         // Crécation d'une instance de la classe Personne
19         Personne personnel = new Personne("John Doe");
20
21         // Accès à l'attribut public et appel de la méthode publique
22         System.out.println("Nom de la personne : " + personnel.nom);
23         personnel.afficherNom();
24     }
25 }

```

50

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Les éléments de l'approche objet

Encapsulation - Java

Attributs et Méthodes Publics :

```

public class Personne {
    // Cette ligne déclare une classe appelée "Personne".
    public String nom;
    // Cette ligne déclare un attribut public appelé "nom". Cela signifie que "nom" peut être directement accédé et modifié depuis l'extérieur de la classe.
    public Personne(String nom) {
        // Cette ligne déclare un constructeur pour la classe "Personne". Ce constructeur accepte un paramètre de type "String" appelé "nom" et initialise l'attribut "nom" de l'objet courant avec la valeur de "nom" passé en paramètre.
        public void afficherNom() {
            // Cette ligne déclare une méthode publique appelée "afficherNom". Cette méthode est utilisée pour afficher le nom de la personne.
            System.out.println("Nom : " + nom);
            // Cette ligne affiche le nom de la personne en utilisant la valeur de l'attribut "nom" de l'objet courant.
        }
    }
}

```

51

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Les éléments de l'approche objet

Encapsulation - Java

Attributs et Méthodes Publics :

```

public class ProgrammePrincipal {
    // Cette ligne déclare une autre classe appelée "ProgrammePrincipal". Cette classe contient la méthode "main" qui est le point d'entrée du programme.
    public static void main(String[] args) {
        // Cette ligne déclare la méthode "main", qui sera exécutée lorsque le programme est lancé.
        Personne personnel = new Personne("Ahmed Ali");
        // Cette ligne crée une instance de la classe "Personne" appelée "personnel" en utilisant le constructeur qui accepte un paramètre "nom". L'objet est ainsi créé avec le nom « Ahmed Ali ».
        System.out.println("Nom de la personne : " + personnel.nom);
        // Cette ligne affiche le nom de la personne en accédant directement à l'attribut public "nom" de l'objet "personnel".
        personnel.afficherNom();
        // Cette ligne appelle la méthode "afficherNom" de l'objet "personnel", ce qui affiche également le nom de la personne, mais en utilisant la méthode.
    }
}

```

52

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Les éléments de l'approche objet

Encapsulation - Java

Attributs et Méthodes Publics :

- Dans l'instruction `Personne personnel = new Personne(" Ahmed Ali");` deux choses se passent :
- La création de l'objet : L'expression `new Personne(" Ahmed Ali")` crée un nouvel objet de la classe `Personne`. Cet objet est une instance de la classe `Personne` avec un nom initialisé à « `Ahmed Ali` » en utilisant le constructeur de la classe qui accepte un paramètre "nom". Cet objet est créé en mémoire et possède ses propres attributs et méthodes.
- L'affectation de l'objet à la variable : L'objet créé est ensuite affecté à la variable `personnel`. Cela signifie que la variable `personnel` maintenant référence (ou pointe vers) l'objet de la classe `Personne` que nous avons créé.
- Utiliser la variable `personnel` pour accéder à cet objet et appeler ses méthodes ou accéder à ses attributs.
 - `System.out.println("Nom de la personne : " + personnel.nom);`
 - `personnel.afficherNom();`

53

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Les éléments de l'approche objet

Encapsulation - Java

Attributs et Méthodes Privés :

```

1  public class CompteBancaire {
2      // Attribut privé
3      private double solde;
4
5      // Constructeur
6      public CompteBancaire(double soldeInitial) {
7          solde = soldeInitial;
8      }
9
10     // Méthode privée pour effectuer un retrait
11     private void retirer(double montant) {
12         if (montant <= solde) {
13             solde -= montant;
14             System.out.println("Retrait de " + montant + " effectué. Solde restant : " + solde);
15         } else {
16             System.out.println("Fonds insuffisants.");
17         }
18     }
19
20     // Méthode publique pour effectuer un retrait en utilisant la méthode privée
21     public void effectuerRetrait(double montant) {
22         retirer(montant);
23     }
24 }
```

54

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Les éléments de l'approche objet

Encapsulation - Java

Attributs et Méthodes Privés :

```

public class Main {
    public static void main(String[] args) {
        // Créez une instance de CompteBancaire
        CompteBancaire compte = new CompteBancaire(1000.0);

        // Appelz la méthode effectuerRetrait avec un montant
        compte.effectuerRetrait(500.0);
    }
}
```

55

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Les éléments de l'approche objet

Encapsulation - Java

Attributs et Méthodes Protégés :

```

1  public class Animal {
2      // Attribut protégé
3      protected String nom;
4
5      // Constructeur
6      public Animal(String nom) {
7          this.nom = nom;
8      }
9
10     // Méthode protégée pour émettre un son
11     protected void emettreSon() {
12         System.out.println(nom + " émet un son.");
13     }
14
15     public class Chien extends Animal {
16         public Chien(String nom) {
17             super(nom);
18         }
19
20         public void aboyer() {
21             emettreSon(); // Accès à la méthode protégée de la classe parente
22             System.out.println(nom + " aboie.");
23         }
24     }
25 }
```

56

Les éléments de l'approche objet

Encapsulation - Java

Attributs et Méthodes Paquetage Privés :

```

1 package monpackage;
2
3 public class Voiture {
4     // Attribut paquetage privé
5     String modèle;
6
7     // Constructeur
8     public Voiture(String modèle) {
9         this.modèle = modèle;
10    }
11
12     // Méthode paquetage privée pour démarrer la voiture
13     void démarrer() {
14         System.out.println("La voiture de modèle " + modèle + " démarre.");
15     }
16 }
17
18 package monpackage;
19
20 public class ProgrammePrincipal {
21     public static void main(String[] args) {
22         Voiture maVoiture = new Voiture("SUW");
23
24         // Accès à l'attribut et à la méthode paquetage privés
25         System.out.println("Modèle de la voiture : " + maVoiture.modèle);
26         maVoiture.démarrer();
27     }
28 }

```

57

Exercices:

- Expliquez la différence entre une classe et un objet en utilisant un exemple concret de votre choix.
- Pour la classe Voiture, identifiez les attributs et les méthodes :
 - marque : String
 - vitesseActuelle : int
 - +démarrer() : void
 - +accélérer(vitesse : int) : void
 - +freiner() : void
 - modèle : String
 - +Couleur
- Expliquez la différence entre les modificateurs de visibilité suivants en UML : - , #, +
- Écrivez le code Java pour créer deux instances différentes de la classe Voiture définie dans l'exercice 2.
- Pour une classe 'CompteBancaire', décrivez trois états différents possibles pour un objet de cette classe.
- Créez un diagramme de classe montrant une relation d'héritage entre une classe 'Véhicule' et deux sous-classes de votre choix.
- Créez une classe 'Employé' qui hérite de la classe 'Personne' et ajoutez-lui un attribut 'salaire'.

58

Correction

1. Classe : C'est un modèle ou un plan qui définit les attributs et les méthodes communs à un ensemble d'objets.

Objet : C'est une instance spécifique d'une classe.

Classe : Voiture (le concept général)

Objets : maVoiture (Renault Clio rouge, immatriculée 123AB45) taVoiture (Peugeot 208 bleue, immatriculée 789XY67)

2.

Attributs :

marque (type String, visibilité privée)

modèle (type String, visibilité privée)

vitesseActuelle (type int, visibilité privée)

Méthodes :

démarrer() (retour void, visibilité publique)

accélérer(vitesse : int) (retour void, visibilité publique)

freiner() (retour void, visibilité publique)

3.

+ : Public - accessible depuis n'importe quelle classe

- : Privé - accessible uniquement à l'intérieur de la classe

: Protégé - accessible dans la classe et ses sous-classes

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Correction

4.

```
public class Voiture {
```

// Attributs (propriétés)

```
private String marque; // Marque de la voiture
```

```
private String modèle; // Modèle de la voiture
```

```
private String couleur; // Couleur de la voiture
```

```
private int vitesseActuelle; // Vitesse actuelle de la voiture
```

// Méthodes (sans implémentation)

```
public void démarrer(); // Démarrer la voiture
```

```
public void accélérer(int vitesse); // Accélérer à une nouvelle vitesse
```

```
public void freiner(); // Freiner la voiture
```

```
}
```

Voiture
-marque: String
-vitesseActuelle: int
-modèle: String
+Couleur: String
+démarrer() : void
+accélérer(vitesse: int) : void
+freiner() : void

59

60

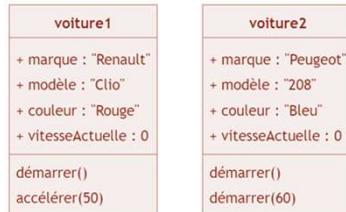
Correction

4.

```
public class TestVoiture {
    public static void main(String[] args) {
        Voiture voiture1 = new Voiture("Renault", "Clio", "Rouge");
        Voiture voiture2 = new Voiture("Peugeot", "208", "Bleu");

        voiture1.démarrer();
        voiture2.démarrer();

        voiture1.accélérer(50);
        voiture2.accélérer(30);
    }
}
```

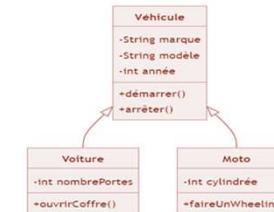


61

Correction

5. Attribut: Solde
 Solde positif :
 $\text{solde} > 0$ peut effectuer des retraits
 Solde nul : $\text{solde} = 0$ limité pour les opérations
 Solde négatif : $\text{solde} < 0$ des frais peuvent s'appliquer

6.



62

Correction

7.

```
classDiagram
    class Personne {
        -String nom
        -String prénom
        -int âge
        +sePresenter()
    }
    class Employé {
        -double salaire
        +calculerSalaireMensuel()
    }
    Employé <|-- Personne

```

63

Les éléments de l'approche objet

Polymorphisme

Le polymorphisme représente la faculté d'une méthode à pouvoir s'appliquer à des objets de classes différentes. Le polymorphisme augmente la générnicité, et donc la qualité, du code.

- Le polymorphisme est un concept important, bien qu'il ne soit pas directement représenté dans les diagrammes de classes ou d'objets, car il est principalement une caractéristique de la programmation orientée objet dans la mise en œuvre.
- Il existe deux types principaux de polymorphisme : le **polymorphisme de surcharge** et le **polymorphisme redéfinition**.

64

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Les éléments de l'approche objet

Polymorphisme

1. Redéfinition (ou Substitution) : est un concept lié à l'héritage en programmation orientée objet.

- La substitution fait référence à la capacité d'une classe dérivée (ou sous-classe) de fournir une implémentation spécifique d'une méthode qui est déjà définie dans sa classe parente (ou superclasse).
- La substitution signifie que vous pouvez utiliser des sous-classes à la place de leur classe parente, mais pas l'inverse.

65

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Les éléments de l'approche objet

Polymorphisme - java

- Redéfinition (ou Substitution)**

L'annotation `@Override` est utilisée en Java pour indiquer explicitement qu'une méthode dans une classe enfant (ou sous-classe) est destinée à remplacer une méthode de la classe parente (ou superclasse) avec la même signature.

```

// Méthode manger() de la classe Animal
public void manger() {
    System.out.println("L'animal mange.");
}

// Définition de la classe Chien qui hérite de la classe Animal
class Chien extends Animal {
    // Redéfinition de la méthode manger() pour la classe Chien
    @Override
    public void manger() {
        System.out.println("Le chien mange de la viande.");
    }
}

// Définition de la classe Chat qui hérite de la classe Animal
class Chat extends Animal {
    // Redéfinition de la méthode manger() pour la classe Chat
    @Override
    public void manger() {
        System.out.println("Le chat mange du poisson.");
    }
}

// Classe principale ExemplePolymorphisme
public class ExemplePolymorphisme {
    public static void main(String[] args) {
        // Crédit d'un objet de type Animal, référence de type Animal
        Animal monAnimal = new Chien();
        monAnimal.manger(); // Appel de la méthode manger() du chien (polymorphisme)

        // Réassignation à un objet de type Chat, même référence de type Animal
        monAnimal = new Chat();
        monAnimal.manger(); // Appel de la méthode manger() du chat (polymorphisme)
    }
}

```

66

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Les éléments de l'approche objet

Polymorphisme

2. Surcharge (Overloading) : La surcharge est un mécanisme qui permet à une classe d'avoir plusieurs méthodes portant le même nom, mais ayant des listes de paramètres différentes.

- En d'autres termes, vous pouvez définir plusieurs méthodes avec le même nom dans une classe, mais elles doivent avoir un nombre différent de paramètres, des types de paramètres différents ou les deux).

La surcharge est utilisée pour fournir plusieurs versions d'une méthode avec un comportement différent en fonction des arguments passés lors de l'appel.

67

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Les éléments de l'approche objet

Polymorphisme - java

- Surcharge (Overloading)**

```

class Calculatrice {
    // Méthode addition prenant deux entiers en paramètre
    public int addition(int a, int b) {
        return a + b;
    }

    // Méthode surchargée pour l'addition de trois entiers
    public int addition(int a, int b, int c) {
        return a + b + c;
    }

    // Méthode surchargée pour l'addition de deux doubles
    public double addition(double a, double b) {
        return a + b;
    }
}

public class ExempleSurcharge {
    public static void main(String[] args) {
        Calculatrice calculatrice = new Calculatrice();

        // Appel de la méthode addition avec deux entiers
        int resultat1 = calculatrice.addition(5, 3);
        System.out.println("Résultat 1 : " + resultat1); // Affiche 8

        // Appel de la méthode addition avec trois entiers
        int resultat2 = calculatrice.addition(5, 3, 2);
        System.out.println("Résultat 2 : " + resultat2); // Affiche 12

        // Appel de la méthode addition avec deux doubles
        double resultat3 = calculatrice.addition(2.5, 3.7);
        System.out.println("Résultat 3 : " + resultat3); // Affiche 6.2
    }
}

```

68

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Les éléments de l'approche objet

Polymorphisme -UML

```

classDiagram
    class Animal {
        +crier() : void
    }
    class Chien {
        +crier() : void
    }
    class Chat {
        +crier() : void
    }
    class Oiseau {
        +crier() : void
    }
    class Calculatrice {
        +additionner(int a, int b) : int
        +additionner(double a, double b) : double
        +additionner(int a, int b, int c) : int
    }
    Animal <|-- Chien
    Animal <|-- Chat
    Animal <|-- Oiseau
    Chien --> Animal : +crier()
    Chat --> Animal : +crier()
    Oiseau --> Animal : +crier()
  
```

69

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Les éléments de l'approche objet

Abstraction

- L'abstraction est l'un des concepts fondamentaux qui permettent de représenter des systèmes et des modèles de manière simplifiée et générale.
- Elle consiste à définir des classes, des méthodes ou des opérations avec des signatures (c'est-à-dire des noms, des paramètres et des types de retour) sans fournir d'implémentation concrète de ces méthodes.
- L'idée est de créer une structure générale qui peut être étendue et spécialisée par les classes dérivées.

70

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Les éléments de l'approche objet

Abstraction

- L'abstraction est utilisée pour masquer les détails inutiles et se concentrer sur les aspects essentiels d'un système.
 - Méthodes abstraites.**
 - Classes abstraites.**
 - Interfaces**

71

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Les éléments de l'approche objet

Abstraction

- Méthodes abstraites:**
 - En programmation orientée objet (POO) et en conception orientée objet (COO), une méthode abstraite est une méthode déclarée dans une classe (ou interface) mais qui n'a pas d'implémentation concrète dans cette classe. Au lieu de cela, la méthode est marquée comme abstraite, ce qui signifie que les sous-classes de cette classe (ou les classes qui implémentent l'interface) sont tenues de fournir une implémentation concrète de cette méthode.
 - Une opération est dite abstraite lorsqu'on connaît sa signature mais pas la manière dont elle peut être réalisée (en UML, on écrit son nom en italique).

72

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Les éléments de l'approche objet

Abstraction

- Classes abstraites:**

- Une classe est dite abstraite lorsqu'elle définit au moins une méthode abstraite ou lorsqu'une classe parent contient une méthode abstraite non encore réalisée (en UML, on écrit son nom en italique).
- Il appartient aux classes enfant de définir les opérations abstraites.
- Le rôle des classes abstraites est de servir de modèle ou de base pour d'autres classes (les sous-classes), qui doivent fournir des implémentations concrètes pour les méthodes abstraites de la classe parente. Par conséquent, la création d'objets se fait généralement à partir des sous-classes, une fois que ces sous-classes ont fourni des implémentations pour toutes les méthodes.

73

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Les éléments de l'approche objet

Abstraction

- Classes abstraites:**

- Il est impossible d'instancier directement une classe abstraite.

```
Forme maForme = new Forme(); // Cela générera une erreur de compilation si Forme est une classe abstraite.
```

- La principale raison pour laquelle vous ne pouvez pas instancier directement une classe abstraite est que les classes abstraites peuvent contenir des méthodes abstraites. Lorsqu'une classe contient une ou plusieurs méthodes abstraites, elle est considérée comme incomplète, et il n'y a donc pas de moyen sensé de créer un objet à partir d'une telle classe, car il manquerait des implémentations concrètes pour ces méthodes.
- Vous实例iez des objets à partir de ces sous-classes, pas à partir de la classe abstraite elle-même.

```
Cercle maForme = new Cercle();
```

74

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Les éléments de l'approche objet

Abstraction - UML

- Le nom de la classe en italique ou entre guillemets ("<nom>"). Une autre manière est d'ajouter « <> » devant le nom de la classe.
- UML, il est couramment accepté d'écrire le nom de la classe de manière normale, mais de mettre en italique le nom de la méthode abstraite

```

classDiagram
    class Forme {
        <<abstract>>
        +calculerSurface() : double
    }
    class Cercle {
        -rayon double
        +calculerSurface() : double
    }
    class Rectangle {
        -longueur double
        -largeur double
        +calculerSurface() : double
    }
    Forme <|-- Cercle
    Forme <|-- Rectangle
  
```

75

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Les éléments de l'approche objet

Abstraction- java

Classe abstraite

```

1 // Définition d'une classe abstraite
2 abstract class Forme {
3     private String nom;
4
5     public Forme(String nom) {
6         this.nom = nom;
7     }
8
9     // Méthode abstraite (doit être implémentée par les sous-classes)
10    public abstract double calculerAire();
11
12    // Méthode concrète
13    public void afficherNom() {
14        System.out.println("Nom de la forme : " + nom);
15    }
16 }
17
18 // Classe concrète qui étend la classe abstraite
19 class Rectangle extends Forme {
20     private double longueur;
21     private double largeur;
22
23     public Rectangle(String nom, double longueur, double largeur) {
24         super(nom);
25         this.longueur = longueur;
26         this.largeur = largeur;
27     }
28
29     @Override
30     public double calculerAire() {
31         return longueur * largeur;
32     }
33 }
34
35 // Classe principale
36 public class ExempleClasseAbstraite {
37     public static void main(String[] args) {
38         Rectangle rectangle = new Rectangle("Mon Rectangle", 3.0, 3.0);
39         rectangle.afficherNom();
40         System.out.println("Aire du rectangle : " + rectangle.calculerAire());
41     }
42 }
  
```

76

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Les éléments de l'approche objet

Abstraction- java

Classe abstraite

```
// Définition d'une classe abstraite:
abstract class Forme {
    private String nom;

    // Constructeur de la classe abstraite:
    public Forme(String nom) {
        this.nom = nom;
    }

    // Méthode abstraite (doit être implémentée par les sous-classes):
    public abstract double calculerAire();

    // Méthode concrète:
    public void afficherNom() {
        System.out.println("Nom de la forme :" + nom);
    }
    // Fermeture de la classe abstraite
}
```

Classe concrète qui étend la classe abstraite:

```
class Rectangle extends Forme {
    private double longueur;
    private double largeur;

    // Constructeur de la classe Rectangle:
    public Rectangle(String nom, double longueur, double largeur) {
        super(nom);
        // Initialisation des attributs spécifiques de cette classe toujours dans ce constructeur:
        this.longueur = longueur;
        this.largeur = largeur;
    }

    // Implémentation de la méthode abstraite calculerAire() de la classe Forme:
    @Override
    public double calculerAire() {
        return longueur * largeur;
    }

    // Classe principale:
    public class ExempleClasseAbstraite {
        public static void main(String[] args) {
            // Création d'un objet de type Rectangle avec un nom, une longueur et une largeur:
            Rectangle rectangle1 = new Rectangle("Mon Rectangle", 5.0, 3.0);

            // Appel de la méthode afficherNom() de l'objet rectangle:
            rectangle1.afficherNom();

            // Calcul de l'aire du rectangle à l'aide de la méthode calculerAire() de l'objet rectangle:
            double aireRectangle = rectangle1.calculerAire();

            // Affichage de l'aire du rectangle:
            System.out.println("Aire du rectangle :" + aireRectangle);
        }
    }
}
```

77

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Les éléments de l'approche objet

Abstraction

- Interface**
- Ancienne définition : « Une interface est essentiellement une collection de méthodes abstraites (non implémentées) qui peuvent être implémentées par des classes qui les utilisent.
- Une interface ne peut contenir que des signatures de méthodes abstraites, c'est-à-dire des méthodes déclarées sans corps (implémentation).
- Définition mise à jour: Depuis Java 8, « une interface en Java est une collection de méthodes (signatures) pouvant inclure à la fois des méthodes abstraites (à implémenter) et des méthodes avec implémentation par défaut (méthodes par défaut). »

78

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Les éléments de l'approche objet

Abstraction

- Interface**
- Une classe peut implémenter plusieurs interfaces en même temps, ce qui permet d'ajouter des comportements spécifiques à différentes interfaces.
- Une Classe peut implémenter plusieurs interfaces, mais elle ne peut hériter que d'une seule classe parente.
- Les interfaces peuvent également contenir des constantes (variables finales) qui sont implicitement statiques et finales.

79

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Les éléments de l'approche objet

Abstraction

- Interface**
- En UML, Une interface est définie comme une classe, avec les mêmes compartiments. On ajoute le stéréotype <> interface >> avant le nom de l'interface.
- On utilise une relation de type réalisation entre une interface et une classe qui l'implémente (flèche d'héritage en pointillés).
- Les classes implémentant une interface doivent implémenter toutes les opérations décrites dans l'interface.

```

classDiagram
    class Calculable {
        +calculer() : void
        +afficher() : void
        +info() : void
    }
    class Cercle {
        -rayon: double
        +calculer() : void
        +afficher() : void
    }
    class Rectangle {
        -longueur: double
        -largeur: double
        +calculer() : void
        +afficher() : void
    }
    Calculable "Implements" Cercle
    Calculable "Implements" Rectangle

```

80

Les éléments de l'approche objet

Abstraction - java

- Interface

```

1 // Définition d'une interface
2 interface MonInterface {
3     void methodeAbstraite(); // Méthode abstraite
4     int VALEUR = 42; // Constante (variable finale et statique)
5 }
6
7 // Une classe qui implemente l'interface
8 class MaClasse implements MonInterface {
9     @Override
10    public void methodeAbstraite() {
11        System.out.println("Implémentation de methodeAbstraite()");
12    }
13
14
15 public class ExempleInterface {
16     public static void main(String[] args) {
17         MaClasse maClasse = new MaClasse();
18         maClasse.methodeAbstraite(); // Appel de la méthode de l'interface
19         System.out.println("Valeur de la constante : " + MonInterface.VALEUR);
20     }
21 }

```

81

Les éléments de l'approche objet

Abstraction- Classe abstraite vs Interface

	Classe abstraite	Interface
Heritage	Les classes abstraites peuvent être étendues (héritées) par d'autres classes. Une classe enfant étend une classe abstraite pour hériter de ses propriétés et méthodes.	Les interfaces sont implémentées par des classes. Une classe implémente une interface pour fournir une implémentation des méthodes définies par cette interface. Une classe peut implémenter plusieurs interfaces en même temps, mais elle ne peut hériter que d'une seule classe parente.
Etat de l'objet	Les classes abstraites peuvent avoir des variables d'instance et des constructeurs, ce qui signifie qu'elles peuvent avoir un état interne (des attributs) et des comportements (des méthodes) liés à cet état.	Les interfaces ne peuvent pas avoir de variables d'instance ni de constructeurs. Elles ne définissent que des méthodes (avec ou sans implémentation par défaut) et ne contiennent pas d'état interne, mais elle peut contenir des constantes.
Objectif	Les classes abstraites sont utilisées lorsque vous souhaitez fournir une base commune pour un groupe de classes liées par l'héritage. Elles sont souvent utilisées pour définir des méthodes abstraites que les sous-classes doivent implémenter.	Les interfaces sont utilisées pour définir des contrats (ensemble de méthodes) que les classes qui les implémentent doivent respecter. Elles sont utilisées pour permettre à des classes non liées par l'héritage d'adopter un comportement commun.
Flexibilité	Les classes abstraites offrent une plus grande flexibilité en termes d'ajout de nouvelles méthodes ou de modification des méthodes existantes sans perturber les classes filles existantes.	Les interfaces offrent une flexibilité accrue en permettant à différentes classes d'implémenter plusieurs interfaces, ce qui permet d'adopter divers comportements.

83

Les éléments de l'approche objet

Abstraction.java

- Interface

```

// Définition d'une interface:
interface MonInterface {
    // Déclaration d'une méthode abstraite:
    void methodeAbstraite();
    // Déclaration d'une constante (variable finale et statique):
    int VALEUR = 42;
}

// Une classe qui implemente l'interface:
class MaClasse implements MonInterface {
    @Override
    public void methodeAbstraite() {
        System.out.println("Implémentation de methodeAbstraite()");
    }
}

```

```

// Classe principale:
public class ExempleInterface {
    public static void main(String[] args) {
        // Création d'une instance de MaClasse:
        MaClasse maClasse1 = new MaClasse();

        // Appel de la méthode de l'interface à partir de l'objet maClasse:
        maClasse1.methodeAbstraite();

        // Accès à la constante VALEUR définie dans l'interface
        System.out.println("Valeur de la constante: "
            + MonInterface.VALEUR);
    }
}

```

82

Les éléments de l'approche objet

Abstraction - Visibilité

- En Java, les méthodes abstraites déclarées dans une interface sont implicitement publiques (public), c'est-à-dire qu'elles sont accessibles depuis n'importe quelle classe qui implémente l'interface. Cela signifie que vous ne pouvez pas déclarer explicitement une méthode abstraite dans une interface comme privée (private) ou protégée (protected).
- En revanche, dans une classe abstraite, les méthodes abstraites peuvent être déclarées avec n'importe quel modificateur d'accès, notamment public, protégé ou même package-private (par défaut). Cependant, il est courant de déclarer ces méthodes comme publiques, car elles sont destinées à être implémentées par les sous-classes qui doivent y accéder.

84

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Les éléments de l'approche objet

L'agrégation et la composition

- L'agrégation et la composition sont deux concepts en programmation orientée objet (POO) qui décrivent les relations entre les objets et les classes dans un système logiciel. Ces concepts sont utilisés pour modéliser la manière dont les objets sont liés les uns aux autres.

85

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Les éléments de l'approche objet

Agrégation :

- L'agrégation est une relation de type « ensemble / élément » entre deux classes, où une classe (l'agrégeateur) contient ou utilise un autre objet (l'agrégré) comme une partie de son état.
- L'objet agrégé peut exister indépendamment de l'objet agrégateur. Lorsque l'objet agrégateur est détruit, l'objet agrégé peut continuer à exister.
- Une relation d'agrégation est généralement représentée par une flèche creuse (avec un losange ouvert) pointant de l'agrégeateur vers l'agrégré dans un diagramme de classes UML.

```

classDiagram
    class Equipe {
        -String nom
        +ajouterJoueur(Joueur joueur)
        +retirerJoueur(Joueur joueur)
    }
    class Joueur {
        -String nom
        -int numero
        +getNom() : String
        +getNumero() : int
    }
    Equipe "3..>" Joueur : contient
  
```

86

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Les éléments de l'approche objet

Composition

- La composition est une relation plus forte que l'agrégation, où l'objet enfant (la partie) est une partie intégrante de l'objet parent (le tout).
- L'objet enfant ne peut pas exister indépendamment de l'objet parent. Lorsque l'objet parent est détruit, l'objet enfant est également détruit.
- Une relation de composition est généralement représentée par une flèche pleine (avec un losange plein) pointant de l'objet parent vers l'objet enfant dans un diagramme de classes UML.

```

classDiagram
    class Maison {
        -String adresse
        -List chambres
    }
    class Chambre {
        -int numero
        -double surface
    }
    class Dossier {
        -String nom
        -List fichiers
    }
    class Fichier {
        -String nom
        -double taille
    }
    Maison "4..>" Chambre : 
    Dossier "4..>" Fichier :
  
```

87

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Les éléments de l'approche objet

La modularité

La capacité de diviser un système complexe en modules ou composants plus petits et indépendants, de manière à simplifier la conception, la compréhension et la gestion du système. UML propose plusieurs diagrammes et concepts qui peuvent être utilisés pour représenter la modularité dans un modèle logiciel.

```

classDiagram
    class Système
    class ModuleA {
        +fonction1()
        +fonction2()
    }
    class ModuleB {
        +fonction3()
        +fonction4()
    }
    class ModuleC {
        +fonction5()
    }
    class ModuleD {
        +fonction6()
    }
    Système --> ModuleA
    Système --> ModuleB
    Système --> ModuleC
    Système --> ModuleD
  
```

88

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Diagramme de classes

Définition

Un diagramme de classes est un type de diagramme UML qui décrit la structure d'un système en montrant les classes du système, leurs attributs, leurs opérations et les relations entre elles.

Le diagramme de classes est considéré comme le plus important de la modélisation orientée objet, il est obligatoire lors d'une telle modélisation.

89

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Diagramme de classes

Objectif

- Déterminer les données qui seront manipulées par le système:
- Donner la structure statique de ces données.
- Représenter les relations statiques existant entre les différentes données du système

90

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Diagramme de classes

Association

Une association est une abstraction de liens qui peuvent exister entre les instances de plusieurs classes

- Dans le monde réel, les objets sont liés physiquement ou bien fonctionnellement les uns avec les autres;
- Ces liens entre objets se traduisent au niveau des classes par des associations;
- Une association représente donc une relation structurelle statique entre deux ou plusieurs classes.

91

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Diagramme de classes

Association

- Association est représentée au moyen d'un trait orienté reliant les classes
- Nommer l'association et préciser les rôles tenus par chaque classe
- Le nom de l'association est en général une forme verbale qui décrit le lien
- Le nom doit apparaître sur l'association (pas attaché à l'une des extrémités). (Une association est souvent nommée par une expression verbale)
- Exemple

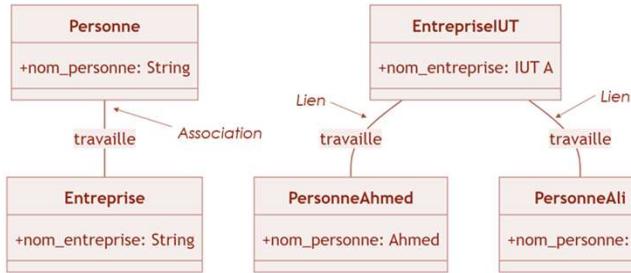
```

classDiagram
    class Personne {
        +nom: String
        +prenom: String
    }
    class Entreprise {
        +nom: String
        +adresse: String
    }
    Personne "1" --> "2" Entreprise : emploie
  
```

92

Diagramme de classes

Association

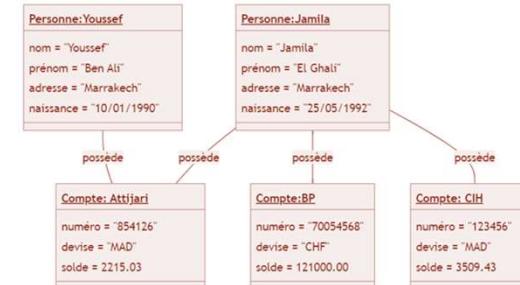


93

Diagramme de classes

Association

Des liens dans diagramme d'objets

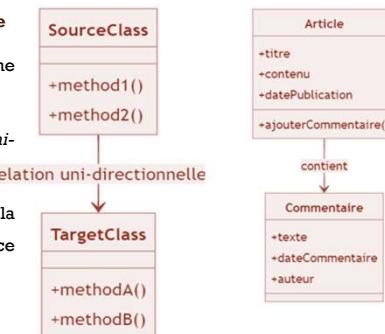


94

Diagramme de classes

La navigabilité : Association uni-directionnelle

- Le trait entre les classes est dirigé par une flèche
- La flèche indique ici que la relation est *uni-directionnelle*.
- La flèche pointe de la classe source vers la classe cible. Cela signifie que la classe source connaît la classe cible, mais pas l'inverse.

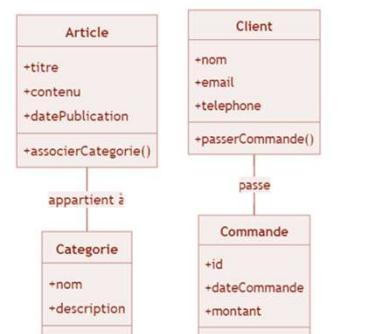


95

Diagramme de classes

La navigabilité : Association bi-directionnelle

- Dans une association bidirectionnelle, il n'y a pas de flèche. Les deux classes connaissent l'existence de l'autre.
- L'absence de flèche indique ici que l'on peut accéder aux catégories à partir des articles qui leur sont liés, et inversement.



96

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Diagramme de classes

Association

Il est possible d'ajouter le sens de lecture du verbe caractérisant l'association sur un diagramme de classe UML, afin d'en faciliter la lecture. On ajoute pour cela un signe < ou > (ou un triangle noir) à côté du nom de l'association.

```

classDiagram
    class Personne {
        +nom
        +prenom
        +email
    }
    class Entreprise {
        +nom
        +adresse
        +secteur
    }
    Personne "Travaille Pour" > Entreprise
  
```

97

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Diagramme de classes

Le rôle d'une association

Il est possible de préciser le rôle joué par une ou plusieurs des classes composant une association afin d'en faciliter la compréhension. On ajoute pour cela ce rôle à côté de la classe concernée (parfois dans un petit encadré collé au trait de l'association).

- Un "rôle" peut être spécifié pour une extrémité de l'association.
- Il exprime le rôle d'une classe dans l'association.
- Il facilite la lecture et la compréhension du modèle objet.

```

classDiagram
    class Classe {
        nom
        role1
        role2
    }
    class Classe
    Personne "Employé" --> "Travaille Pour" --> "Employeur" Entreprise
  
```

98

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Diagramme de classes

Multiplicités de l'association

Les multiplicités permettent de contraindre le nombre d'objets intervenant dans les instanciations des associations.

(Définir le nombre d'occurrences d'objets d'une classe qui peuvent être associées à des objets d'une autre classe dans une relation)

99

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Diagramme de classes

Multiplicités de l'association

On en place de chaque côté des associations.

- Une multiplicité d'un côté spécifie combien d'objets de la classe du côté considéré sont associés à un objet donné de la classe de l'autre côté.
- Syntaxe : min..max, où min et max sont des nombres représentant respectivement les nombres minimaux et maximaux d'objets concernés par l'association.
- Pour le lire simplement, on parcourt la relation depuis une des classes de la relation, vers une autre classe. On peut s'aider en disant : "Une instance (une occurrence, un exemplaire...) de la première classe, peut être liée (reliee par la relation en question) à au minimum x et au maximum y instances de l'autre classe".

100

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Diagramme de classes

Multiplicités de l'association

Certaines écritures possibles :

- $0..*$ = "au minimum 0 et au maximum un nombre quelconque",
- $*$ = "au minimum 0 et au maximum un nombre quelconque (équivalent de $0..*$)",
- $1..1$ = "au minimum 1 et au maximum 1", donc "exactement 1",
- 1 = "au minimum 1 et au maximum 1" (l'équivalent de $1..1$),
- $1..*$ = "au minimum 1 et au maximum un nombre quelconque",
- $2..7$ = "au minimum 2 et au maximum 7",
- $x..y$ = d'une manière générale "au minimum x et au maximum y".

101

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Diagramme de classes

Multiplicités de l'association

Exemples

```

classDiagram
    class Article {
        +id
        +titre
        +contenu
    }
    class Categorie {
        +id
        +nom
    }
    Article "*" --> "1..5" Categorie : appartient à
  
```

- Ici, le $1..5$ s'interprète comme: à un objet donné de la classe Article, on doit associer au minimum 1 objet de la classe Categorie et on peut en associer au maximum 5.
- Catégorie peut avoir zéro ou plusieurs articles.
- (la multiplicité doit être cohérente avec la réalité du domaine métier).

102

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Diagramme de classes

Multiplicités de l'association

Exemples:

```

classDiagram
    class Societe {
        +nom
        +adresse
    }
    class Client {
        +nom
        +prenom
    }
    class Personne {
        +nom
        +prenom
        +poste
    }
    class Voiture {
        +marque
        +modele
    }
    Societe "*" --> "1" Personne : emploie
    Client "0..1" --> "*" Voiture : achete
  
```

103

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Diagramme de classes

Multiplicités de l'association

Exemple:

- Une société emploie de « une » à plusieurs personnes.
- Une personne appartient à une seule société.
- Un client achète zéro à plusieurs voitures.
- Une voiture peut être achetée par un seul client.

```

classDiagram
    class Societe {
        +nom
        +adresse
    }
    class Client {
        +nom
        +prenom
    }
    class Personne {
        +nom
        +prenom
        +poste
    }
    class Voiture {
        +marque
        +modele
    }
    Societe "*" --> "1" Personne : emploie
    Client "0..1" --> "*" Voiture : achete
  
```

104

Brahim Jaber ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Diagramme de classes

Multiplicités de l'association

Exemple:

```

classDiagram
    class Etudiant {
        +nom
        +prenom
        +dossier
    }
    class EcoleIngenieur {
        +nom
        +adresse
    }
    Etudiant "0..1" -- "1..*" EcoleIngenieur : candidaté à
  
```

105

Brahim Jaber ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Diagramme de classes

Multiplicités de l'association

Exemple:

```

classDiagram
    class Etudiant {
        +nom
        +prenom
        +dossier
    }
    class EcoleIngenieur {
        +nom
        +adresse
    }
    Etudiant "0" -- "1..*" EcoleIngenieur : candidaté à
  
```

Un étudiant a fait au moins une demande de poursuite d'études mais peut aussi en avoir fait plusieurs
Une école d'ingénier peut ne pas avoir de candidats, mais peut en avoir plusieurs.

106

Brahim Jaber ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Diagramme de classes

Les relations entre classes en JAVA: « one to many »

Les tableaux et les listes en Java peuvent tous deux être utilisés pour représenter une relation "one to many" entre deux classes, où un élément d'une classe est associé à plusieurs éléments de l'autre classe.

Les listes sont généralement plus flexibles et appropriées pour ce type de relation en raison de leur capacité à redimensionner dynamiquement et à fournir des méthodes de gestion plus pratiques.

107

Brahim Jaber ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Diagramme de classes

Les relations entre classes en JAVA : « one to many »

Tableaux :

Les tableaux ont une taille fixe déterminée lors de leur création. Par conséquent, ils conviennent si vous connaissez à l'avance le nombre maximum d'éléments que vous devez stocker.

En Java, pour définir un tableau, vous pouvez utiliser l'opérateur new, comme vous l'avez montré dans votre exemple :

```

int[] unTableau = new int[10];
  
```

int[] : C'est le type de données du tableau, indiquant que ce tableau contiendra des valeurs entières (nombres entiers).

unTableau : C'est le nom du tableau que vous avez choisi. Vous pouvez utiliser n'importe quel nom valide pour votre tableau.

new int[10] : C'est l'opérateur new utilisé pour allouer de l'espace mémoire pour le tableau. [10] indique que le tableau aura une taille de 10 éléments, allant de l'indice 0 à l'indice 9.

108

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Diagramme de classes

Les relations entre classes en JAVA : « one to many »

Listes:

Les listes sont des collections dynamiques qui peuvent redimensionner automatiquement pour s'adapter à de nouveaux éléments. Elles n'ont pas de taille fixe et sont appropriées pour des relations "one to many".

```
import java.util.List;
import java.util.ArrayList;

public class ExempleListe {
    public static void main(String[] args) {
        // Déclaration d'une liste d'entiers
        List<Integer> listeEntiers = new ArrayList<>();
        //...
    }
}
```

109

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Diagramme de classes

Les relations entre classes en JAVA : « one to many »

Il est également possible de ne pas initialiser explicitement la liste dans la classe et de l'initialiser en dehors de la classe à travers les méthodes d'accès (getters) et de modification (setters).

Exemple:

```
public class Client {
    private String nom;
    private List<Facture> factures;
}
```

La classe Facture est destinée à être utilisée pour stocker des objets de type Facture, il est logique de déclarer la liste factures dans la classe Client en tant que liste d'objets Facture.

110

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Diagramme de classes

Les relations entre classes en JAVA : « one to many »

Association de « un » à *.

```
public class ClasseA {
    // one to many
    private List<ClasseB> NotreListe;
    // Autres attributs de la classe A

    public List<ClasseB> getListeB() {
        return NotreListe;
    }

    public void setListeB(List<ClasseB> NotreListe) {
        this.NotreListe = NotreListe;
    }

    // Autres membres de la classe A
}
```

La déclaration `List<ClasseB> NotreListe` indique que `NotreListe` est une liste (ou collection) qui stocke plusieurs objets de type `ClasseB`. Ces objets de type `ClasseB` sont associés à un objet de type `ClasseA`.

111

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Diagramme de classes

Les relations entre classes en JAVA : « many to one »

Association de * à un.

```
public class ClasseB {
    // many to one
    private ClasseA leA;
    // Autres attributs et méthodes de la classe B

    public ClasseB(type attribut1, type attribut2, ClasseA leA) {
        this.attribut1 = attribut1;
        this.attribut2 = attribut2;
        this.leA = leA;
    }
}
```

Private `ClasseA leA;` dans la classe `ClasseB` représente une relation "many to one" entre la classe `ClasseB` et la classe `ClasseA`. Cela signifie qu'un objet de la classe `ClasseB` est associé à un et un seul objet de la classe `ClasseA`.

112

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Diagramme de classes

Les relations entre classes en JAVA :

Navigabilité restreinte. (Association unidirectionnelle)

```
public class A {
    private List<ClasseB> NotreListe;
    // Autres attributs la classe A
}

public class B {
    // pas de référence à un objet de la classe A
}
```

113

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Diagramme de classes

Les relations entre classes en JAVA :

Exemple:

```
import java.util.ArrayList;
import java.util.List;
```

```
public class Client {
    private int id;
    private String nom;
    private String email;
    private List<Commande> commandes; // Relation one to many avec Commande

    public Client(int id, String nom, String email) {
        this.id = id;
        this.nom = nom;
        this.email = email;
    }

    public List<Commande> getCommandes() {
        return commandes;
    }

    public void setCommandes(List<Commande> commandes) {
        this.commandes = commandes;
    }

    // Autres membres de la classe Client
}

public class Commande {
    private int numero;
    private String date;
    private Client client; // Relation many to one avec Client

    public Commande(int numero, String date, Client client) {
        this.numero = numero;
        this.date = date;
        this.client = client;
    }

    // Autres membres de la classe Commande
}
```

114

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Diagramme de classes

Les relations entre classes en JAVA :One to One

Exemple:

NB: Il faut savoir si une relation un-à-un entre deux classes devrait être modélisée sous forme d'attribut plutôt que d'une relation d'association.

```
//One to One
public class Personne {
    private String nom;
    private CarteIdentite carteIdentite;

    public Personne(String nom, CarteIdentite carteIdentite) {
        this.nom = nom;
        this.carteIdentite = carteIdentite;
    }

    // Getters et setters
}

public class CarteIdentite {
    private String numero;
    private String dateExpiration;

    public CarteIdentite(String numero, String dateExpiration) {
        this.numero = numero;
        this.dateExpiration = dateExpiration;
    }
}
```

115

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Diagramme de classes

Dimension d'une association

La dimension d'une association se réfère au nombre d'entités (classes) qu'elle relie. Plus précisément, la dimension d'une association est le nombre d'extrémités de l'association, c'est-à-dire le nombre de classes ou d'entités qui participent à cette association.

- Une association binaire relie deux entités, donc elle a une dimension de 2.
- Une association N-aire est une relation entre trois classes ou plus (ternaire ou plus), c'est-à-dire des associations de plus grande dimension.
- Une association réflexive est une association qui relie une classe à elle-même, elle est donc de dimension 1.

116

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Diagramme de classes

Dimension d'une association

Association binaire est représentée par une ligne qui relie les deux classes et peut être dirigée ou non dirigée. Par exemple, une association binaire peut exister entre les classes "Client" et "Commande", où un client peut passer une ou plusieurs commandes.

```

classDiagram
    class Societe {
        +nom
        +adresse
    }
    class Client {
        +nom
        +prenom
    }
    class Personne {
        +nom
        +prenom
        +poste
    }
    class Voiture {
        +marque
        +modele
    }

    Client "1" --> "1.." Personne : emploie
    Client "0..1" --> "*" Voiture : achete
  
```

117

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Diagramme de classes

Dimension d'une association

Association n-aire : Elle est représentée par un losange qui relie les classes et peut avoir plusieurs traits qui partent du losange pour se connecter aux classes. Les associations n-aires sont souvent utilisées pour représenter des relations complexes entre plusieurs classes.

```

classDiagram
    class Salle {
        +String numero
        +int capacite
    }
    class Etudiant {
        +String nom
        +String numeroEtudiant
    }
    class Professeur {
        +String nom
        +String specialite
    }
    class Cours {
        +String intitule
        +String code
    }

    Salle "1" --- "2..*" Etudiant : 
    Salle "1" --- "1" Professeur : 
    Etudiant "2..*" --> "1" Cours : 
  
```

118

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Diagramme de classes

Dimension d'une association

Une association réflexive est une association entre une classe et elle-même. Elle est parfois appelée "auto-association". Une telle association modélise une relation ou une connexion entre les instances d'une classe.

Dans l'exemple, l'association « Ami » est une relation réflexive au sein de la classe "Personne". Chaque instance de la classe "Personne" peut avoir des amis.

```

classDiagram
    class Personne {
        +nom
        +prenom
    }

    Personne "*" --> "0..*" Personne : est ami de
  
```

119

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Diagramme de classes

Dimension d'une association

Auto-association dans les associations réflexives:

Ici, chaque instance de la classe peut être associée à zéro, une, ou plusieurs autres instances de la même classe, y compris à elle-même.

- Si l'on souhaite exprimer le contraire (une instance peut être associée avec d'autres instances de la même classe, mais pas avec elle-même) : on ajoute une contrainte en UML

```

classDiagram
    class Personne {
        +nom
        +prenom
    }

    Personne "*" --> "0..*" Personne : est ami de
  
```

120

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Diagramme de classes

Dimension d'une association

Le nommage des rôles est essentiel à la clarté du diagramme.

Exemple:

```

classDiagram
    class Personne {
        +nom
        +prenom
    }
    class Employe
    Personne "1" -- "*" Employe : Chef
  
```

Encadre >

121

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Diagramme de classes

Récapitulatif

```

classDiagram
    class Personne {
        nom : string
        prénom : string
        adresse : string
        naissance : Date
        ouvrirCompte(init : float)
    }
    class Compte {
        numéro : int
        devise : Devise
        solde : float
        déposer(montant : float)
        retirer(montant : float)
        solde() : float
    }
    Personne "1..2" -- "*" Compte : sesPropriétaires
    Personne "*" -- "1..2" Compte : possède "Sens"
    Note over Personne: nom de l'association (optionnel)
  
```

Lien = instance d'association

122

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Diagramme de classes

Plusieurs associations entre deux classes

Il est autorisé d'associer deux classes par plusieurs associations:

```

classDiagram
    class Personne
    class Avion
    class Voiture
    Personne "1..2" -- "1..2" Avion : pilote, Passager
    Personne "*" -- "4" Voiture : Conduire>, Démarrer>, Posséder>, Laver>
  
```

123

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Diagramme de classes

Contraintes, invariants

Dans un diagramme de classe en UML, les contraintes sont utilisées pour spécifier des règles, des conditions qui s'appliquent aux éléments du diagramme, tels que les classes, les attributs, les opérations, la multiplicité et les associations.

Expression de ces contraintes:

- Dans le diagramme lui-même.
- Dans des commentaires ou des notes
- Un document séparé
- Langages dédiés comme OCL (Object Constraint Language)

124

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Diagramme de classes

Contraintes sur les attributs

sous forme de note

```

classDiagram
    class Produit {
        id : int {unique}
        prixHT : real {prixHT ≥ 0}
        taxe : real
        prixTTC : real
    }
    note over Produit: 
        prixTTC = prixHT × (1 + taxe)
        0 ≤ taxe ≤ 1
    
```

dans le diagramme

Contraintes sur la classe Produit :

- L'identifiant est unique
- Le prix hors taxe est toujours positif ou nul
- La taxe est comprise entre 0 et 1
- Le prix TTC est égal au prixHT augmenté de la valeur de la taxe appliquée au prixHT

dans un document annexe

125

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Diagramme de classes

Contraintes sur les attributs- Attribut dérivé

- Un attribut dérivé dans un diagramme de classe UML est un attribut dont la valeur est calculée à partir d'autres attributs de la même classe.
- Il est important de noter que la valeur de l'attribut dérivé n'est pas stockée dans la base de données, mais est calculée à la volée à partir des autres attributs.
- L'attribut dérivé doit être signalé en préfixant le nom de l'attribut avec une barre oblique « / »

```

classDiagram
    class Personne {
        nom
        adresse
        dateNaissance
        /âge
    }
    class AttributDerve
    
```

126

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Diagramme de classes

Contraintes sur une association

```

classDiagram
    class Salle {
        +String nom
        +int capacité
    }
    class Place {
        +int numéro
    }
    Salle "1" -- "1..*" Place : "nombre de places associées à une salle = capacité"
    
```

contrainte sur une association

nombre de places associées à une salle = capacité

127

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Diagramme de classes

Contraintes sur une association

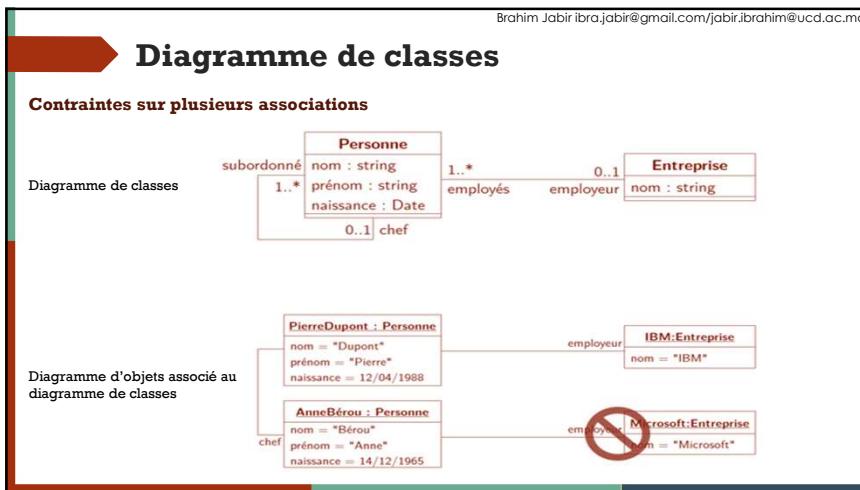
Les contraintes peuvent s'exprimer en langage naturel. Graphiquement, il s'agit d'un texte encadré d'accolades.

```

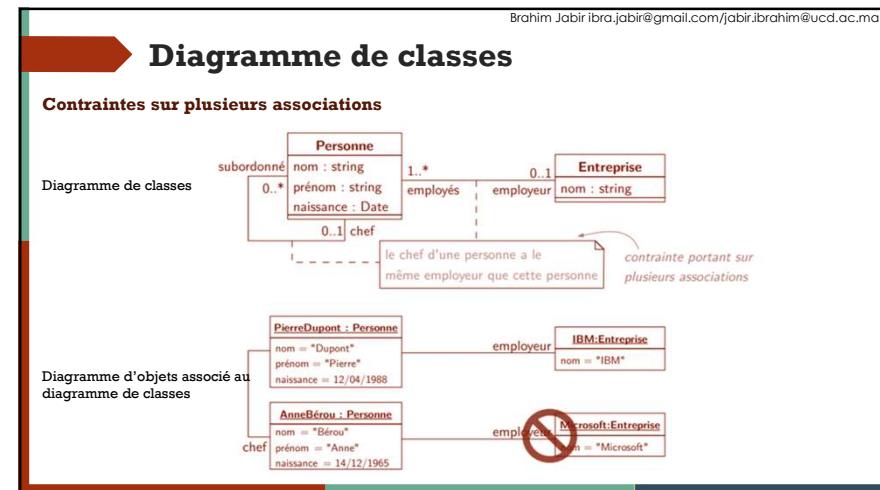
classDiagram
    class Universite
    class Personne
    Universite "0..1" -- "1..n" Personne : étudiant
    Universite "*" -- "1..n" Personne : enseignant
    {ou}
    
```

Ici, on indique qu'une personne joue soit le rôle d'étudiant, soit le rôle d'enseignant pour une université donnée (« ou » exclusif).

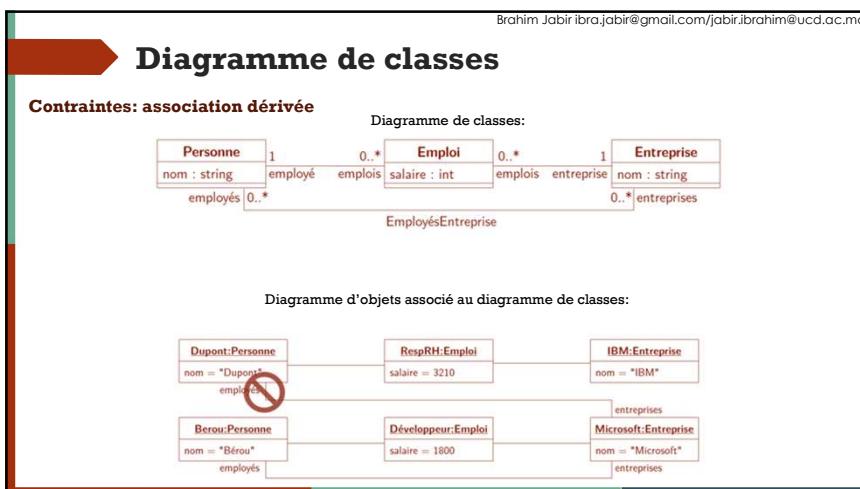
128



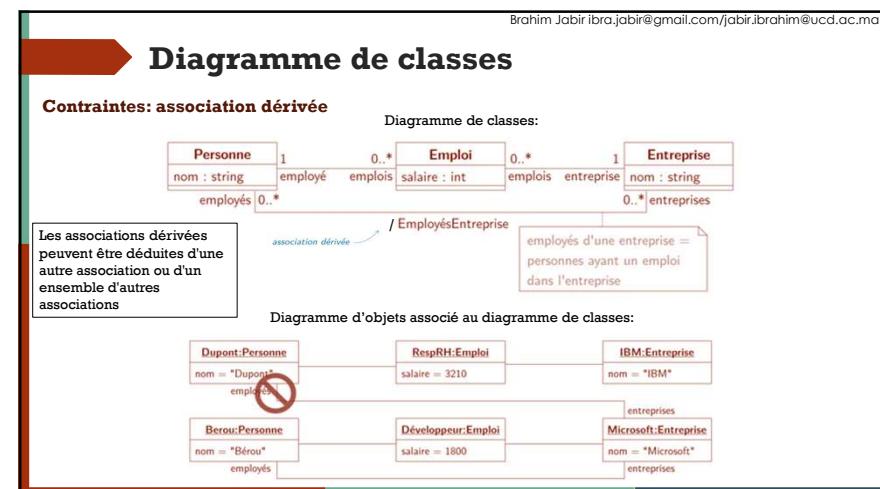
129



130



131



132

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Diagramme de classes

Contraintes sur la multiplicité

Une contrainte est généralement ajoutée entre accolades {} près de la notation de la multiplicité.

{ordered} : Cette contrainte indique que les objets de la classe associée doivent être ordonnés dans la collection d'éléments associés.

un père ne peut pas être lui-même : {self.parents <> self}

```

classDiagram
    class Personne {
        nom : string
        prenom : string
        naissance : Date
    }
    Personne "2" -- "*" Personne : parents
    Personne "*" -- "<{ordered}" Personne : enfants
    note over Personne : ensemble ordonné
  
```

133

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Diagramme de classes

Contraintes sur la multiplicité

L'annotation {bag} est généralement utilisée lorsque vous souhaitez spécifier que plusieurs occurrences du même type d'objet peuvent être incluses dans une collection. (doublons autorisés).

L'ordre n'est pas significatif.

```

classDiagram
    class Commande {
        numero : int
        total : float
        date : Date
    }
    Commande "1..*" -- "1..*" Pizza : {bag}
    note over Pizza : multi-ensemble  
 (chaque élément peut apparaître plusieurs fois)
  
```

134

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Diagramme de classes

Classe d'association

L'utilisation d'une classe-association permet d'ajouter des attributs, des opérations ou d'autres informations spécifiques à cette relation.

Graphiquement, on la relie à l'association avec des pointillés.

```

classDiagram
    class Etudiant
    class Sujet
    class Examen {
        +Date date
    }
    Etudiant "*" -- "*" Sujet : Examen
  
```

135

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Diagramme de classes

Classe d'association

```

classDiagram
    class Livre {
        +int numeroExemplaire
        +String titre
        +Date dateParution
    }
    class Emprunter {
        +Date dateDebut
        +Date dateFin
    }
    class Abonne {
        +int numeroDeCarte
        +String nomAbonne
        +String adresseAbonne
    }
    Livre "0..*" -- "0..*" Emprunter
    Emprunter "0..*" -- "0..*" Abonne
  
```

136

Brahim Jabir ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Diagramme de classes

Équivalence de la Classe d'association

Parfois, l'information véhiculée par une classe-association peut être véhiculée d'une autre manière:

```

classDiagram
    class A
    class B
    class C
    class Livre {
        +int numeroExemplaire
        +String titre
        +Date dateParution
    }
    class Abonne {
        +int numeroDeCarte
        +String nomAbonne
        +String adresseAbonne
    }
    class Emprunter {
        +Date dateDebut
        +Date dateFin
    }

    A "n" --> "m..n" C
    C --> "1..n" B
    C --> "1..n" Livre
    C --> "1..n" Abonne
    Livre "*" --> "*" Emprunter
    Abonne "*" --> "*" Emprunter
  
```

Deux modélisations modélisant la même information.

137

Brahim Jabir ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Diagramme de classes

Classe d'association: implémentation java: many to many

```

public class Livre {
    private int num;
    private String titre;
    private Date dateParution;
    // Autres attributs de Livre
}

public class Abonne {
    private List<Emprunter> emprunts;
    private int num;
    private String nom;
    private String adresse;
    // Autres attributs d'Abonne
}

public class Emprunter {
    private Livre livre;
    private Abonne abonne;
    private Date dateDebut;
    private Date dateFin;
    // Autres attributs pour l'emprunt
    // Autres méthodes pour manipuler les emprunts
}
  
```

```

classDiagram
    class Livre
    class Abonne
    class Emprunter {
        +Livre livre
        +Abonne abonne
        +Date dateDebut
        +Date dateFin
    }

    Livre "*" --> "*" Emprunter
    Abonne "*" --> "*" Emprunter
  
```

138

Brahim Jabir ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Diagramme de classes

Exemple d'implémenter un Diagramme de Classes en java

Réalité

DÉSIGNATION	PU	QTE	SOUS TOTAL
GALAXY S4	1000,00	2	2000,00
GALAXY S5	1500,00	2	3000,00
PC HP 4555	4500,00	1	4500,00
IMPRIMANTE LASER 200	1500,00	1	1500,00
BUREAU PLAX-55	1200,00	1	1200,00

Total : 12000,00

Règlement Espèce

Modèle

139

Brahim Jabir ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Diagramme de classes

Implémenter un Diagramme de Classes en java (Classe client)

```

public class Client {
    private Long id;
    private String nom;

    // Si visibilité
    List<Facture> factures;

    public Long getId() { return id; }

    public void setId(Long id) { this.id = id; }

    public String getNom() { return nom; }

    public void setNom(String nom) { this.nom = nom; }

    public List<Facture> getFactures() { return factures; }

    public void setFactures(List<Facture> factures) { this.factures = factures; }

    public Client(Long id, String nom) {
        this.id = id;
        this.nom = nom;
    }
}
  
```

140

Diagramme de classes
Implémenter un Diagramme de Classes en java (**Classe Facture**)

```

private String numero;
private Date date;
//many to one
private Client client;

//One to many
private List<LigneDeCommande> lignesDeCommande;

public String getNumero() { return numero; }

public void setNumero(String numero) { this.numero = numero; }

public Date getDate() { return date; }

public void setDate(Date date) { this.date = date; }

public List<LigneDeCommande> getLignesDeCommande() { return lignesDeCommande; }

public void setLignesDeCommande(List<LigneDeCommande> lignesDeCommande) {
    this.lignesDeCommande = lignesDeCommande;
}

public Facture(String numero, Date date, Client client) {
    this.numero = numero;
    this.date = date;
    this.client = client;
}

```

141

Diagramme de classes
Implémenter un Diagramme de Classes en java (**Classe d'association LigneCommande**)

```

package ma.application.businessmodel;

public class LigneDeCommande {
    private Long id;
    private Integer qte;
    private Double souTotal;
    private Facture facture;
    private Produit produit;

    public Long getId() { return id; }
}

```

142

Diagramme de classes
Implémenter un Diagramme de Classes en java (**Classe Produit**)

Vous n'avez pas besoin d'une liste de LigneCommande dans la classe Produit. La classe Produit n'a pas besoin de conserver une liste de toutes les lignes de commande auxquelles il est associé, car cette information est généralement stockée du côté de la classe Facture. Vous pouvez accéder aux produits associés à une facture via les objets LigneCommande.

```

public class Produit {
    // Autres attributs et méthodes de la classe Produit
}

```

143

Diagramme de classes
Implémenter un Diagramme de Classes en java (**Classe Main**)

```

import ...

public class Main {

    public static void main(String[] args) {
        Client client = getClient(Long.valueOf(args[0]));
        System.out.println("Factures du client ayant numéro " + client.getId() + " " + client.getNom());
        System.out.println("-----");
        System.out.println("-----");

        List<Facture> facturesClient = getFactures(client);
        for (Facture facture : facturesClient) {
            System.out.println("Numéro " + facture.getNumero());
            System.out.println("Date " + facture.getDate());
            System.out.println("-----");
        }
    }
}

```

La classe utilisée comme point d'entrée de l'application. C'est là où le programme Java commence son exécution. La classe Main contient généralement la méthode main, qui est le point de départ de l'exécution de l'application.

144

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Diagramme de classes

Exercice d'application

Soit le cas "Réservation de vols dans une agence de voyage"

- 1° La compagnie aérienne propose différents vols.
- 2° Un vol est ouvert à la réservation et fermé sur ordre de la compagnie.
- 3° Un client peut réserver un ou plusieurs vols, pour des passagers différents.
- 4° Une réservation peut être annulée ou confirmée.
- 5° Un vol a un aéroport de départ et un aéroport d'arrivée.
- 6° Un vol a un jour et une heure de départ et un jour et une heure d'arrivée.
- 7° Un vol peut comporter des escales dans des aéroports
- 8° Une escale a une heure d'arrivée et une heure de départ.

145

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Diagramme d'objets

Définition

- Les diagrammes d'objets représentent un ensemble d'objets et leurs liens. Ce sont des vues statiques des instances des éléments qui apparaissent dans les diagrammes de classes.
- Un diagramme d'objet est une instance d'un diagramme de classes.
- La création d'un diagramme d'objets nécessite:
 - Identifier les types d'objets que vous souhaitez représenter dans votre diagramme.
 - Créer des objets : Pour chaque classe, créer des instances ou des objets spécifiques que vous souhaitez inclure dans le diagramme et inclure les attributs et les opérations
 - Identifier les liens : les relations entre les objets. Les relations courantes incluent l'agrégation, la composition, l'association, l'héritage, etc.
 - ...
 - Les objets (instances de classes)
 - Les liens (instances d'associations)

146

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Diagramme d'objets

Rôles

Les diagrammes d'objets font partie de la catégorie des diagrammes structurels statiques dans l'UML.

- Il fournit une vue figée du système à un moment précis. Ils capturent l'état actuel des objets et montrent comment ils sont connectés à ce moment précis.
- Il peut se servir pour donner des exemples pour mieux comprendre un diagramme de classes.
- Un diagramme de classes peut en effet correspondre à de nombreux diagrammes d'objets différents.
- Son principal rôle consiste à valider un diagramme de classe surtout pour des situations complexes.

147

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Diagramme d'objets

Représentation d'un objet:

- Rectangle : Chaque objet est représenté par un rectangle, parfois appelé une boîte d'objet. Ce rectangle contient des informations sur l'objet et ses attributs.
- Nom de l'Objet : Le nom de l'objet est généralement affiché en haut du rectangle. Il est souvent souligné pour indiquer qu'il s'agit d'un objet spécifique plutôt que du nom d'une classe.
- Attributs de l'Objet : Si nécessaire, les attributs de l'objet peuvent être listés dans le rectangle, généralement sous le nom de l'objet. Les attributs sont généralement affichés sous la forme "nomAttribut : valeurAttribut". Par exemple, si l'objet est une voiture et a un attribut "Couleur" avec la valeur "Rouge".
- Les méthodes (c'est-à-dire les opérations ou les fonctions) ne sont généralement pas représentées directement sur le diagramme d'objets.

148

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Diagramme d'objets

Représentation d'un objet:

Le diagramme d'objet est une "photo instantanée" qui montre l'état des objets à un moment donné.

nomObjet : nomClasse
nomAttribut1 = valeur1
.....
nomAttributn = valeurn

voiture1
+ marque : "Renault"
+ modèle : "Clio"
+ couleur : "Rouge"
+ vitesseActuelle : 20

voiture2
+ marque : "Peugeot"
+ modèle : "208"
+ couleur : "Bleu"
+ vitesseActuelle : 0

149

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Diagramme d'objets

Représentation d'un objet:

- Notation du nom de l'objet**
- Objet nommé:
nomObjet : nomClasse
- Objet anonyme:
:nomClasse
- Nom de l'objet sans spécifier le nom de la classe:
nomObjet
- Instance multiple:
:nomClasse

150

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Diagramme d'objets

Représentation d'un objet:

- Les attributs**
- Les représentations des objets peuvent contenir des attributs significatifs:

Produit
code
prix_unit
seuil_reapprov
quantité
mettre_a_jour_quantité()

ordinateur : Produit
code = O145
prix_unit = 4800
seuil_reapprov = 100
quantité=50

La classe

L'instance de la classe: Objet

151

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Diagramme d'objets

Représentation d'un objet:

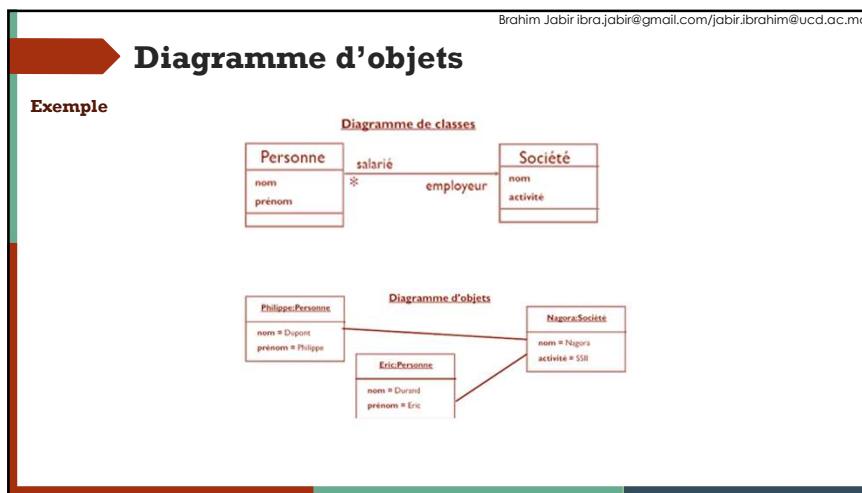
- Les liens**
- Les objets sont reliés par des instances d'associations : les liens.

Facture
numéro
date
totalHT
:Facture
Fc2023-15
16/10/2023
65000

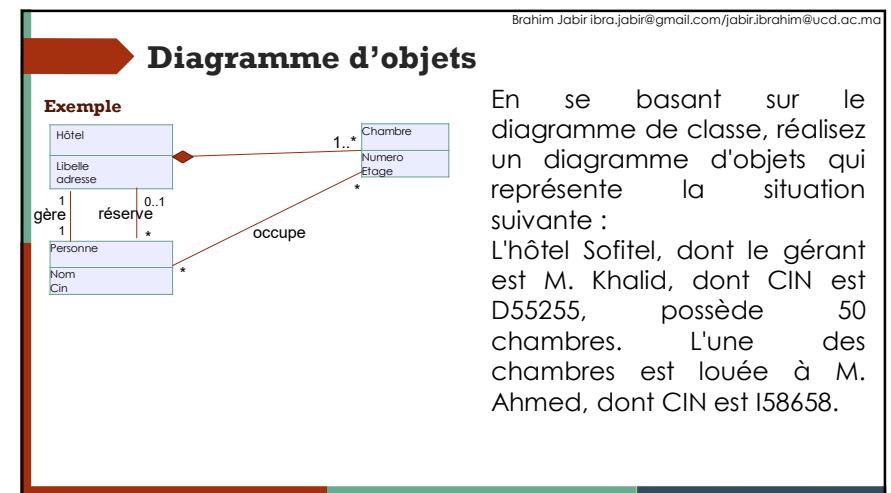
:Facture
Fc2023-16
18/10/2023
56200

Client
codeClient
nomClient
:Client
CI1256
Ahmed_SARL

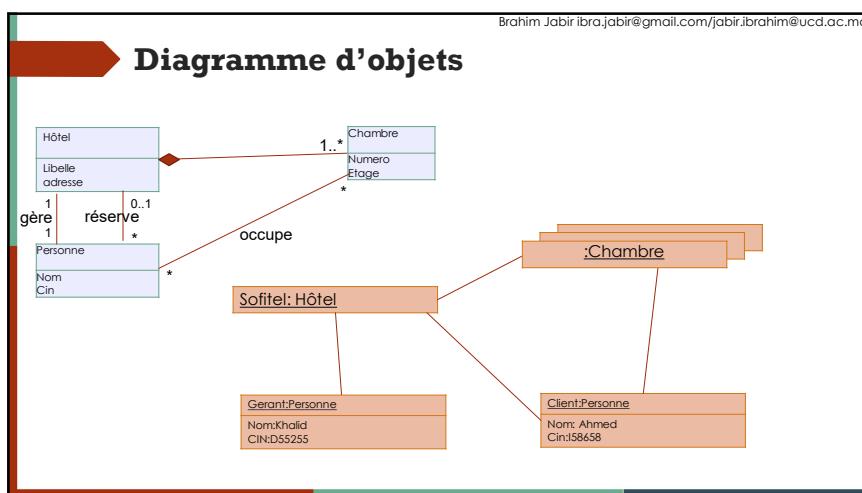
152



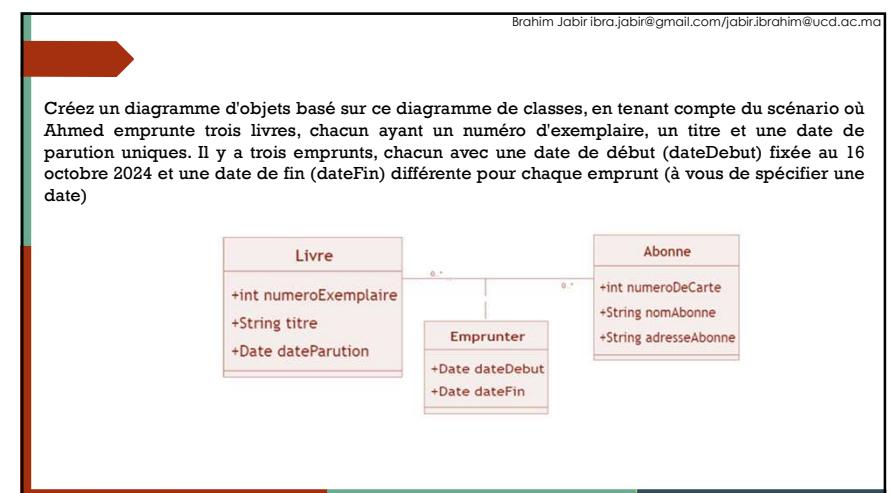
153



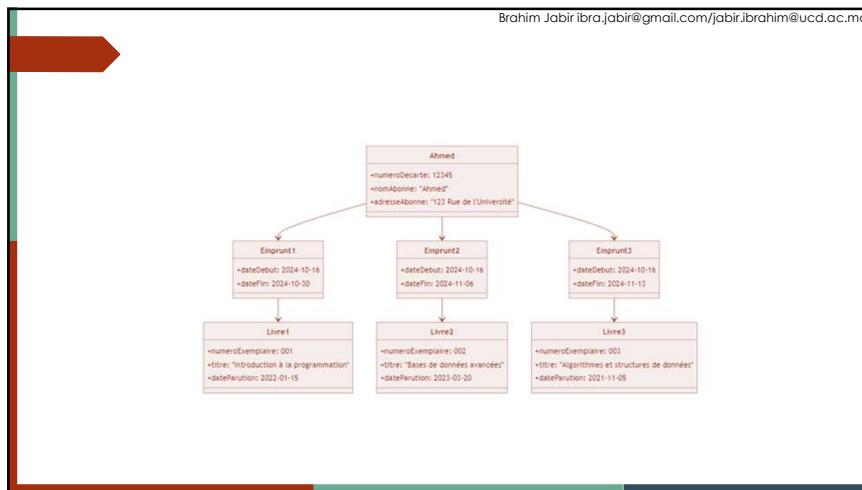
154



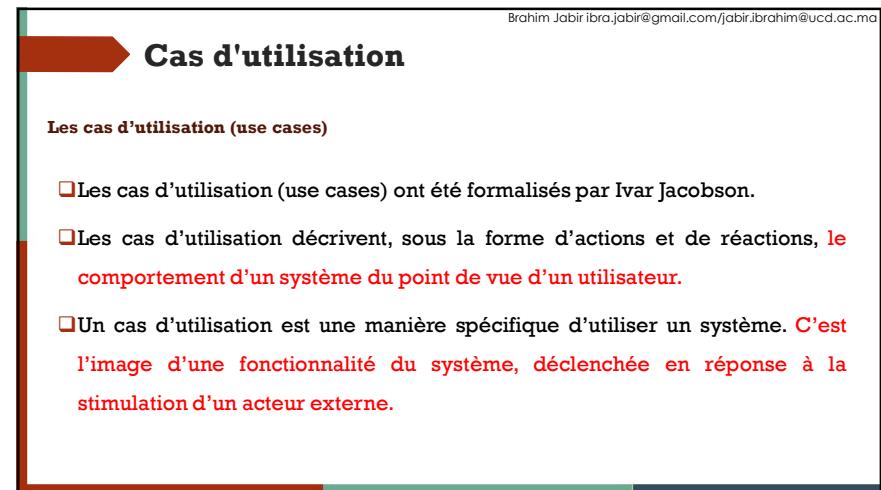
155



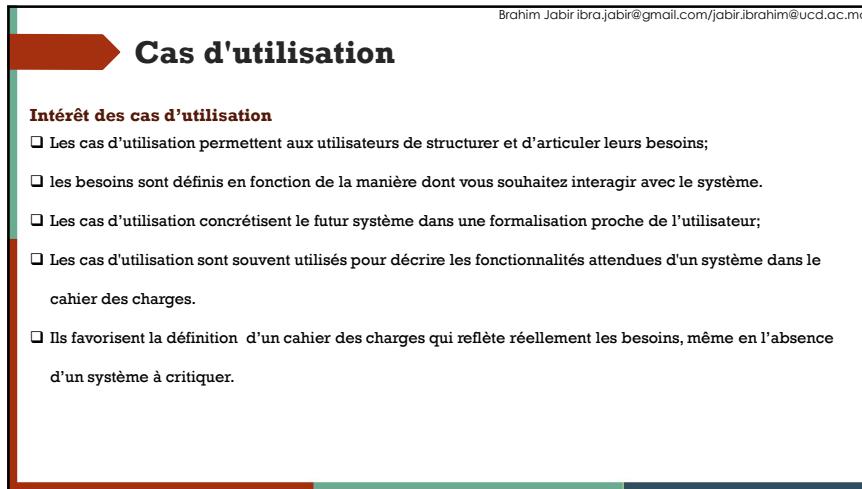
156



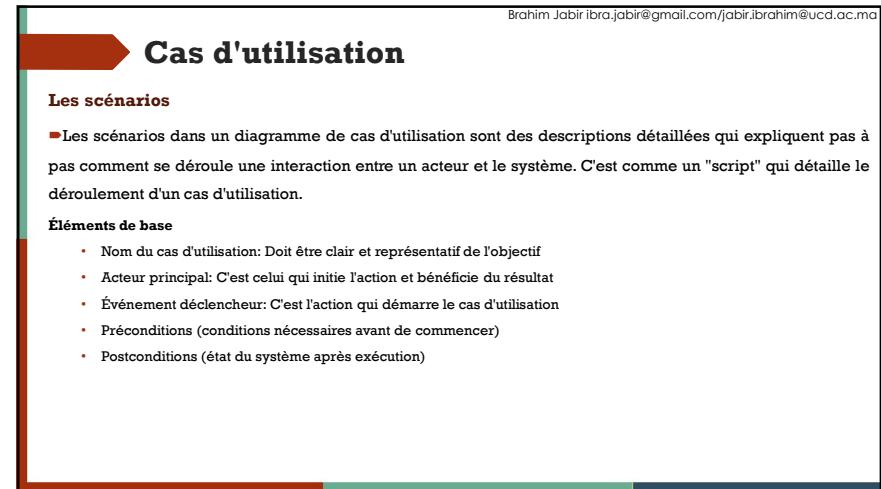
157



158



159



160

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Cas d'utilisation

Les scénarios

Nom du cas d'utilisation

Règles principales :

1. Commencer par un verbe à l'infinitif
 - "Réserver un billet"
 - "Réservation d'un billet"
 - "Consulter son solde"
 - "Consultation du solde"
2. Être concis mais précis
 - "Modifier un profil"
 - "Modifier les informations personnelles du profil utilisateur"
3. Inclure un complément d'objet
 - "Valider une commande"
 - "Valider"
4. Reflèter une fonctionnalité complète
 - "Gérer les clients »
 - "Liste clients"

161

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Cas d'utilisation

Les scénarios

Types de scénarios

- **Scénario nominal** : le déroulement normal, sans erreur
- **Scénarios alternatifs** : les autres chemins possibles
- **Scénarios d'erreur** : gestion des cas d'échec

Exemple:

Cas d'Utilisation : Effectuer un Retrait d'Argent

- **Déclenché par** : Un utilisateur insérant sa carte bancaire et saisissant le montant de retrait.
- **Nom** : Effectuer un Retrait d'Argent

162

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Cas d'utilisation

Les scénarios

Scénario nominal :

- 1.Le client insère sa carte
- 2.Le système demande le code PIN
- 3.Le client entre son code
- 4.Le système vérifie le code
- 5.Le système affiche les options
- 6.Le client choisit "retrait" et entre le montant
- 7.Le système vérifie le solde
- 8.Le système délivre l'argent
- 9.Le système rend la carte

163

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Cas d'utilisation

Les scénarios

Scénario alternatif (solde insuffisant) :

- 1.à 7. (identique au nominal)
- 2.Le système affiche "solde insuffisant"
- 3.Retour à l'étape 5.

Scénario d'erreur (code PIN incorrect) :

- 1.à 3. (identique au nominal)
- 2.Le système indique que le code est incorrect
- 3.Retour à l'étape 2 (3 essais maximum)

164

Cas d'utilisation

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Notation

- Un CU est représenté par un ovale
- Le nom du CU apparaît à l'intérieur de l'ovale.
- Il est composé .
 - 1- Nom de la fonctionnalité qu'il prend en charge

Syntaxe

```

graph LR
    CU([CU]) --- R[Retirer argent]
  
```

165

Cas d'utilisation

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Acteurs - Définition

- Un Acteur définit un rôle qu'une entité extérieure assume lors de son interaction avec le système
- Un acteur représente un utilisateur ou un système externe avec lequel interagit le système en cours de modélisation.
 - L'acteur dialogue par la suite avec le CU dont il est l'initiateur
 - L'acteur possède un nom: rôle qu'il joue lors de son interaction avec le système
 - L'acteur n'est pas forcément humain : autre système (logiciels ; Automates)

166

Cas d'utilisation

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

4 grandes catégories d'acteurs

- Les acteurs principaux: les personnes qui utilisent les fonctions principales du système.
- Les acteurs secondaires: les personnes qui effectuent des tâches administratives ou de maintenance.
- Le matériel externe: les dispositifs matériels incontournables qui font partie du domaine de l'application et qui doivent être utilisés.
- Les autres systèmes: les systèmes avec lesquels le système doit interagir.

167

Cas d'utilisation

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Acteur- Syntaxe

- Symbole : L'acteur est généralement représenté par un simple dessin en forme de "stick figure" ou bonhomme allumette. C'est une silhouette humaine stylisée avec une tête ronde et un corps en trait.
- Nom : Sous la figure, on écrit le nom de l'acteur.
- Connexions : Des lignes droites relient l'acteur aux cas d'utilisation avec lesquels il interagit. Un acteur est souvent associé à plusieurs cas d'utilisation ...

```

graph LR
    Actor(( )) --- CU([Cas d'utilisation])
    subgraph ActorLabel [nom de l'acteur.]
        Actor
    end
  
```

168

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Cas d'utilisation

Acteurs- Exemple 1

- » Mohamed utilise le système pour gérer son agenda »
- » Youssef utilise le système pour gérer son agenda »
- » Ali utilise aussi le système pour gérer son agenda »
- » Mais Ali est aussi autorisé à administrer le système »

Combien d'acteurs?

169

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Cas d'utilisation

Acteurs- Exemple 1

- » Mohamed utilise le système pour gérer son agenda »
- » Youssef utilise le système pour gérer son agenda »
- » Ali utilise aussi le système pour gérer son agenda »
- » Mais Ali est aussi autorisé à administrer le système »
 - Le rôle « Utilisateur » est un acteur du système
 - Le rôle « Administrateur » est un acteur du système

170

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Cas d'utilisation

Acteurs- Exemple 1

```

graph LR
    Utilisateur --> Système
    Administrateur --> Système
    Système --> SA[S'Authentifier]
    Système --> GA[Gérer Agenda]
    Système --> GCU[Gérer Comptes Utilisateurs]
  
```

171

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Cas d'utilisation

Acteurs -Exemple 2

Khalid, garagiste de son état;

- » Passe le plus claire de son temps dans le rôle de Mécanicien, mais peut à l'occasion jouer le rôle de vendeur.
- » Dimanche, il joue le rôle de client et entretient sa voiture personnelle.
- » Combien de rôles sont présents ici, et combien de personnes y a-t-il ?

172

Cas d'utilisation

Brahim Jabir ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Acteurs - Exemple 2

```

graph LR
    Client --> Conduire
    Client --> Entretenir
    Mechanicien --> Reparer
    Vendeur --> Vendre
  
```

The diagram illustrates a UML Use Case Diagram. It features three actors on the left: "Client", "Mécanicien", and "Vendeur". Each actor is connected by a line to one or more use cases listed on the right. The use cases are: "Conduire", "Entretenir", "Réparer", and "Vendre".

173

Cas d'utilisation

Brahim Jabir ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Acteurs - Exemple 2

- Une même personne physique peut jouer le rôle de plusieurs acteurs.
- Ne pas confondre **personne physique** et **rôle**.
- Une personne peut très bien assumer plusieurs rôles et réciproquement.

174

Cas d'utilisation

Brahim Jabir ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Les relations dans un diagramme cas d'utilisation

1. La relation de communication
2. La relation de généralisation
3. La relation d'inclusion
4. La relation d'extension

```

graph TD
    Actor1 -- Communication --> UseCase1
    SubActor -- Généralisation --> SuperActor
    CasSpécifique -- Généralisation --> CasGénéral
    UseCase2 -- "+include+" --> UseCase3
    UseCase4 -- "+extend+" --> UseCase5
  
```

The diagram shows four types of relationships in a UML Use Case Diagram.
 1. **Communication**: An arrow from Actor1 to UseCase1.
 2. **Généralisation**: An arrow from SubActor to SuperActor, and another from Cas spéciique to Cas général.
 3. **Inclusion**: A dashed arrow labeled "+include+" from UseCase2 to UseCase3.
 4. **Extension**: A dashed arrow labeled "+extend+" from UseCase4 to UseCase5.

175

Cas d'utilisation

Brahim Jabir ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

La relation de généralisation

Cette relation de généralisation/spécialisation est présente dans la plupart des diagrammes UML et se traduit par le concept d'héritage dans les langages de programmation orientés objet.

Exemple :

Un directeur est une sorte de commercial : il peut faire avec le système tout ce que peut faire un commercial, plus d'autres choses.

```

graph LR
    Système["Système"]
    Utilisateur
    Directeur
    Administrateur
    SAutentifier("S'Authentifier")
    GAgenda("Gérer Agenda")
    GComptes("Gérer Comptes Utilisateurs")

    Utilisateur --> SAutentifier
    Utilisateur --> GAgenda
    Utilisateur --> GComptes
    Directeur --> SAutentifier
    Directeur --> GAgenda
    Directeur --> GComptes
    Administrateur --> GAgenda
    Administrateur --> GComptes
  
```

The diagram illustrates inheritance relationships.
 - **Generalization**: An arrow from "Commercial" to "Directeur".
 - **Inclusion**: Dashed arrows from "Utilisateur" to "S'Authentifier", "Gérer Agenda", and "Gérer Comptes Utilisateurs".
 - **Extension**: Dashed arrows from "Directeur" to "S'Authentifier", "Gérer Agenda", and "Gérer Comptes Utilisateurs".
 - **Generalization**: Dashed arrows from "Administrateur" to "Gérer Agenda" and "Gérer Comptes Utilisateurs".

176

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Cas d'utilisation

La relation de généralisation

Cette relation de généralisation/specialisation est présente dans la plupart des diagrammes UML et se traduit par le concept d'héritage dans les langages de programmation orientés objet.

Exemple :

Un directeur est une sorte de commercial : il peut faire avec le système tout ce que peut faire un commercial, plus d'autres choses.

177

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Cas d'utilisation

La relation de généralisation

Il y a généralisation entre un cas A et un cas B lorsqu'on peut dire : A est une sorte de B, ou entre acteur A et acteur B.

178

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Cas d'utilisation

La relation d'inclusion

La relation d'inclusion a un caractère obligatoire, la source spécifiant à quel endroit le cas d'utilisation cible doit être inclus. Les cas d'utilisation inclus sont généralement nécessaires pour accomplir le cas d'utilisation principal.

Les Cas liés par inclusion («include») : Sont AUTOMATIQUEMENT déclenchés par le système ne sont pas liés directement à l'acteur. L'utilisateur n'a pas le choix, c'est une étape obligatoire du processus. L'utilisateur n'a pas le choix, c'est une étape obligatoire du processus.

On utilise une ligne en pointillés avec la mention <<include>> pour indiquer cette relation.

179

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Cas d'utilisation

La relation d'inclusion

1. "Retirer argent" et "Imprimer reçu" :
 - Initiés par l'utilisateur (il décide)
 - Lien direct avec l'acteur
2. "Vérifier solde" et "Logger transaction" :
 - Initiés automatiquement par le système
 - Font partie du processus obligatoire
 - Liés par «include»
 - Pas de lien direct avec l'acteur

180

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Cas d'utilisation

La relation d'extension

Dans une relation d'extension entre cas d'utilisation, le cas d'utilisation source ajoute son comportement au cas d'utilisation destination (cible). L'extension peut être soumise à une condition.(gérer des scénarios optionnels ou conditionnels).

Le cas d'extension est déclenché conditionnellement à l'intérieur du système. Cela signifie qu'il peut être activé dans certaines situations spécifiques, mais il n'est pas exécuté systématiquement. Ce cas d'extension est souvent proposé à l'utilisateur comme une option supplémentaire à traiter, mais l'utilisateur peut décider de l'utiliser ou non.

Représentation graphique : La relation d'extension est aussi représentée par une ligne en pointillés, mais avec l'étiquette <>extend><.

```

    graph LR
        Client --> Système_de_paiement[Système de paiement]
        Système_de_paiement -- "include" --> Payer_achat[Payer achat]
        Payer_achat -- "extend" --> Utiliser_coupon[Utiliser coupon]
        Utiliser_coupon --> Système_de_paiement
        note [Extension : pas besoin de lien direct avec l'acteur même si c'est lui qui choisit d'utiliser le coupon]
    
```

181

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Cas d'utilisation

La relation d'extension

- Cas de base "Payer achat" → lié directement au Client (il initie l'action)
- Cas d'extension "Utiliser coupon" → PAS besoin de lien direct avec le Client car :
 - C'est une option du cas de base
 - Il ne peut pas être initié seul, sans le cas de base
 - L'extension représente une variante ou un ajout au scénario principal
- Cas inclus "Vérifier solde" → automatique, pas de lien direct

```

    graph LR
        Client --> Système_de_paiement[Système de paiement]
        Système_de_paiement -- "include" --> Payer_achat[Payer achat]
        Payer_achat -- "extend" --> Utiliser_coupon[Utiliser coupon]
        Utiliser_coupon --> Système_de_paiement
        note [Extension : pas besoin de lien direct avec l'acteur même si c'est lui qui choisit d'utiliser le coupon]
    
```

182

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Cas d'utilisation

La relation d'extension

Dans le cas d'utilisation "Effectuer un virement", si "Vérifier le solde" est exécuté **systématiquement**, mais uniquement lorsque certaines conditions spécifiques sont remplies (comme un montant supérieur à 500 francs), cela relève toujours d'une relation d'extension.

"Vérifier le solde" n'est pas toujours déclenché, mais uniquement dans un contexte spécifique (virement > 500 francs). Cela en fait un **comportement conditionnel**. Même si l'utilisateur ne choisit pas de l'activer, le système le fait en fonction de la situation.

L'**extension** est utilisée pour ajouter ce type de fonctionnalité **conditionnelle**. Le cas principal (ici, "Effectuer un virement") peut s'exécuter sans l'extension dans des circonstances normales, mais lorsque la condition est remplie, l'extension est déclenchée par le système.

```

    graph LR
        Client --> Système_Bancaire[Système Bancaire]
        Système_Bancaire -- "include" --> Effectuer_virement[Effectuer virement]
        Effectuer_virement -- "extend" --> Vérifier_solde[Vérifier solde]
        Vérifier_solde -- "montant ≥ 500 francs" --> Effectuer_virement
        note [EXTEND (conditionnel) :  
- Si la condition est remplie  
- Déclenché par le système  
- L'utilisateur n'a pas le choix  
- Condition : montant > 500 francs]
    
```

183

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Diagramme de cas d'utilisation

Réutilisation de cas d'utilisation

Les relations d'inclusion et d'extension permettent d'isoler un service réutilisable comme partie de plusieurs autres cas d'utilisation :

- On parle alors de **réutilisation**.
- Le code développé pour implémenter le cas d'utilisation réutilisé est d'emblée identifié comme ne devant être développé qu'une seule fois, puis réutilisé.

```

    graph LR
        Lecteur --> Système_de_gestion_de_bibliothèque[Système de gestion de bibliothèque]
        Système_de_gestion_de_bibliothèque -- "include" --> RéserverLivre[RéserverLivre]
        Système_de_gestion_de_bibliothèque -- "include" --> EmprunterLivre[EmprunterLivre]
        Système_de_gestion_de_bibliothèque -- "include" --> RechercherLivre[RechercherLivre]
        note [inclusion]
    
```

184

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Diagramme de cas d'utilisation

Exemple

- Le diagramme des cas d'utilisation comprend les acteurs, le système et les cas d'utilisation eux-mêmes.
- Les acteurs déclenchent les cas d'utilisation.
- Le système de notation est un cadre rectangulaire qui englobe l'ensemble des cas d'utilisation du système.

```

graph LR
    A((Acteur A)) --> X((Cas d'utilisation X))
    B((Acteur B)) --> Y((Cas d'utilisation Y))
    subgraph Système [Système]
        X
        Y
    end
  
```

185

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Exercices

- Rédigez un scénario de cas d'utilisation pour le cas "Emprunter un livre" dans un système de bibliothèque en ligne.
- Créez un scénario de cas d'utilisation pour le cas "Passer une commande" dans une application de commande de repas en ligne.

186

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Exercices

Créez un diagramme de cas d'utilisation complet pour un système de gestion de bibliothèque. Le système doit gérer les fonctionnalités suivantes :

- Inscription des membres
- Recherche de livres
- Emprunt de livres
- Retour de livres
- Renouvellement de prêts
- Réservation de livres
- Gestion des amendes
- Ajout de nouveaux livres au catalogue
- Génération de rapports statistiques

187

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Exercices

Elaborez un diagramme de cas d'utilisation pour un système de réservation de vols en ligne. Le système doit prendre en charge les fonctionnalités suivantes :

- Enregistrement en ligne
- Recherche de vols
- Réservation de vols
- Vérifier la disponibilité(vérification automatique)
- Modification de réservation
- Annulation de réservation
- Paiement des réservations
- Confirmation de paiement (par le système)
- Gestion des remboursements
- Assistance client en ligne

188

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Diagramme de cas d'utilisation

Créez un diagramme de cas d'utilisation représentant les interactions entre le client et le système de commande de repas. Identifiez les principaux acteurs et décrivez leurs interactions avec les différentes fonctionnalités du système.

189

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Diagramme de cas d'utilisation

Exercice d'application

Dans un établissement scolaire, on désire gérer la réservation des salles de cours ainsi que du matériel pédagogique (ordinateur portable ou/et Vidéo projecteur). Seuls les enseignants sont habilités à effectuer des réservations (sous réserve de disponibilité de la salle ou du matériel).

Le planning des salles peut quant à lui être consulté par tout le monde (enseignants et étudiants). Par contre, le récapitulatif (calculé à partir du planning des salles) ne peut être consulté que par les enseignants.

Enfin, il existe pour chaque formation un enseignant responsable qui est le seul qui peut éditer le récapitulatif horaire pour l'ensemble de la formation

190

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Diagramme de séquence

Définition et objectifs

- Décrire dans l'ordre chronologique les interactions entre les objets du système en précisant les contraintes.
- Une interaction se traduit par un envoi de messages entre objets du système.
- Le diagramme de séquence permet de faire apparaître:
 - Les objets intervenant dans l'interaction (acteurs ou objets intervenant dans le système),
 - La description de l'interaction,
 - Les interactions entre les intervenants

191

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Diagramme de séquence

Eléments du diagramme de séquence

- Acteurs
- Objets (instances)
- Messages (cas d'utilisation, appels d'opération)

Principes de base : Représentation graphique de la chronologie des échanges de messages avec le système ou au sein du système

- Échanges de messages représentés horizontalement.
- chaque objet/acteur a une "ligne de vie" verticale qui représente sa durée d'existence

```

sequenceDiagram
    participant A as A:Acteur
    participant o1 as o1:Classe1
    participant o2 as o2:Classe2
    A->>o1: casutilisation()
    activate o1
    o1-->>A: retour
    activate o2
    o1->>o2: operation(args)
  
```

192

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Diagramme de séquence

Les objets

- Les objets représentés sont uniquement ceux participant à la collaboration.
- Un objet est matérialisé par un rectangle et une barre verticale, appelé **ligne de vie** de l'objet (Object lifeline).
- Le Nom de l'objet ou du Rôle est souligné pour indiquer qu'il s'agit d'une instance.*
- 3 représentations sont possibles :
 - Le nom de l'objet uniquement:
 - Le nom de la classe uniquement:
 - Le nom de l'objet et de la classe, séparés par :

193

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Diagramme de séquence

Les messages

- Les objets communiquent en échangeant des messages.
- Un message est représenté par une flèche horizontale, orientée de l'émetteur vers le destinataire.

• Une flèche allant de l'appelant du message spécifie un message dans un diagramme de séquence. Un message peut circuler dans n'importe quelle direction, de gauche à droite, de droite à gauche ou vers l'appelant lui-même. Si la flèche permet de décrire le message envoyé d'un objet à l'autre, les différentes pointes de flèche permettent d'indiquer le type de message envoyé ou reçu.

194

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Diagramme de séquence

L'utilisateur

- Un acteur est représenté afin de traduire les interactions déclenchées par un élément extérieur au système.
- Un acteur déclenche le premier message de l'interaction.
- Un acteur a la même représentation graphique qu'un acteur dans les Cas d'Utilisation.
- Un acteur est un rôle.

195

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Diagramme de séquence

Exemple

Modéliser le calcul du poids d'une voiture (poids moteur + poids carrosserie) par un calculateur.

Commentaire :

Dimension verticale : (le temps) ; L'ordre d'envoi d'un message est déterminé par sa position sur l'axe vertical du diagramme ; le temps s'écoule "de haut en bas" de cet axe : la numérotation des messages est optionnelle.

Dimension horizontale : les objets (et les acteurs) ; La disposition des objets sur l'axe horizontal n'a pas de conséquence pour la sémantique du diagramme.

196

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Diagramme de séquence

Classes spécifiques

Diagramme de cas d'utilisation
Diagramme de classes du système

Objectif : Le diagramme de séquence représente les interactions entre les différents objets du système qui se produisent lors de la réalisation d'un cas d'utilisation spécifique.

- Le diagramme de séquence peut représenter soit un seul cas d'utilisation, soit plusieurs cas d'utilisation liés entre eux.
- le diagramme de séquence ne montre pas nécessairement les interactions entre tous les objets du système, mais seulement ceux qui sont impliqués dans la réalisation du cas d'utilisation spécifique.

Problème : Communication entre les acteurs et le système vu comme un ensemble d'objets.

197

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Diagramme de séquence

Éléments du diagramme de séquence

Diagramme de cas d'utilisation
Diagramme de classes du système
Diagramme d'interface

Solution

Communication entre acteurs et système via une interface (texte, web, physique...)

198

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Diagramme de séquence

Éléments du diagramme de séquence

Diagramme de cas d'utilisation
Diagramme de classes du système
Diagramme d'interface et de contrôle

Solution : Création de classes de contrôle et de classes d'interface qui :

- gèrent les interactions avec les acteurs
- encapsulent le résultat des opérations

199

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Diagramme de séquence

Les objets: stéréotypes

Dans le cadre du développement d'applications, les **stéréotypes d'objets** sont utilisés pour séparer les responsabilités des différentes parties du système.

« Boundary »
« Entity »
« Control »

Ces stéréotypes sont notamment issus de l'architecture **MVC (Modèle-Vue-Contrôleur)**.

200

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Diagramme de séquence

Les objets: stéréotypes

« Boundary » « Entity » « Control »

- ❑ une « frontière » (« boundary » en anglais) encapsule l'interaction avec des acteurs externes (utilisateurs humains ou systèmes externes);
- ❑ une entité représente une information de longue durée qui est pertinente pour les parties prenantes (c'est-à-dire principalement des objets de domaine, et généralement des données persistantes);
- ❑ un « contrôle » assure les traitements requis pour l'exécution d'un cas d'utilisation et de sa logique métier, et coordonne les interactions entre les autres objets impliqués dans le cas d'utilisation.

201

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Diagramme de séquence

Éléments du diagramme de séquence

```

sequenceDiagram
    participant Acteur
    participant Clavier
    participant Ecran
    participant Interface
    participant Contrôle
    participant obj1 as obj1 : Classe1
    participant obj2 as obj2 : Classe2

    Acteur->>Clavier: saisir()
    activate Clavier
    Clavier-->>Interface: opération(args)
    activate Interface
    Interface-->>Contrôle: retour
    activate Contrôle
    Contrôle-->>obj1: Classe2()
    activate obj1
    obj1-->>obj2: obj2
    activate obj2
    obj2-->>Interface: affichage
    deactivate obj2
    Interface-->>Ecran: affichage
    activate Ecran
    Ecran-->>Acteur: affichage
    deactivate Ecran
    deactivate Contrôle
    deactivate obj1
  
```

202

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Diagramme de séquence

Éléments du diagramme de séquence

Il n'est pas obligatoire de présenter systématiquement ces trois types de classes spécifiques (interface, contrôleur, modèle) dans un diagramme de séquence. Cela dépend de :

1. Le niveau de détail souhaité
 - Version simplifiée (sans interface explicite)
 - Version détaillée : peut inclure tous les objets impliqués
2. L'objectif du diagramme
 - Pour une vue d'ensemble : structure simple
 - Pour une documentation technique : plus de détails
 - Pour une analyse système : structure complète MVC
3. Le contexte d'utilisation
 - Phase d'analyse : peut-être plus abstrait
 - Phase de conception : plus détaillé
 - Documentation utilisateur : plus simplifié
4. Bonnes pratiques
 - Inclure uniquement les classes pertinentes pour le scénario
 - Maintenir la lisibilité du diagramme
 - Se concentrer sur les interactions importantes

203

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Diagramme de séquence

Activations et envois de messages

- ❑ Une période d'activité (ou d'activation) (Focus of control) correspond au temps pendant lequel un objet effectue une action, soit directement, soit par l'intermédiaire d'un autre objet qui lui sert de sous-traitant.
- ❑ Les périodes d'activité se représentent par des bandes rectangulaires placées sur les lignes de vie.
- ❑ Le début et la fin d'une bande correspondent respectivement au début et à la fin d'une période d'activité.

Représentation de la période d'activité d'un objet

UnObjet

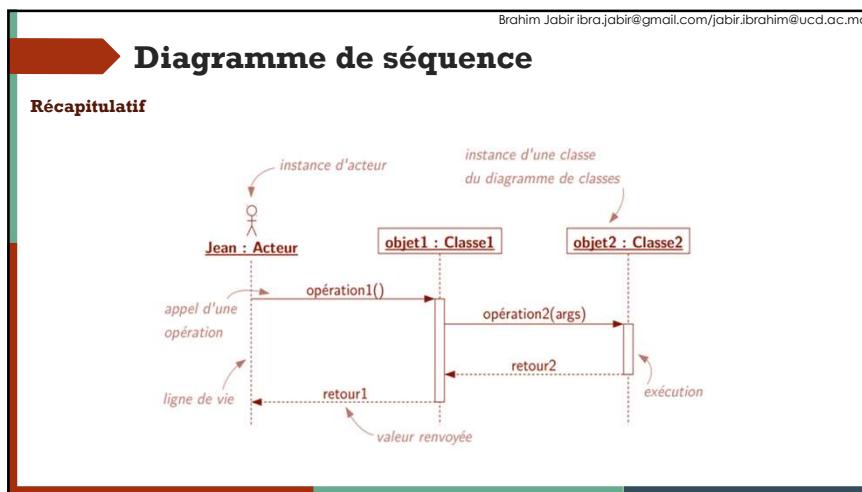
Activation

durée d'activation

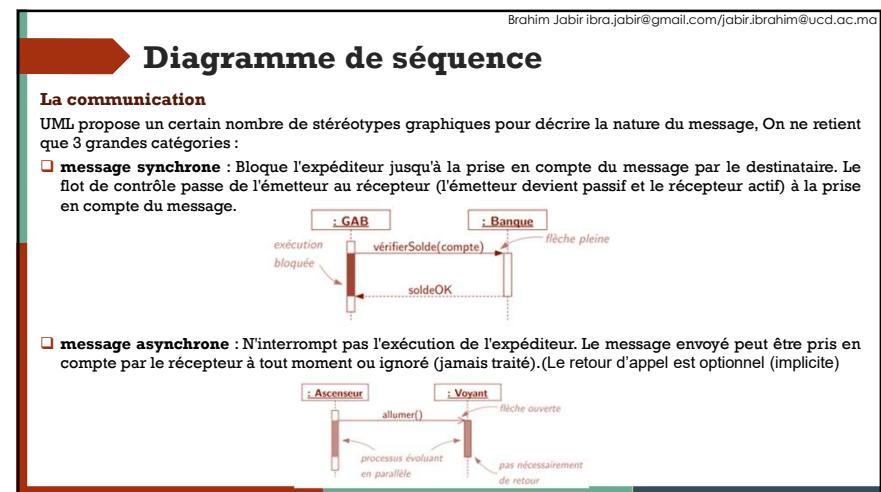
Remarque :

- ❑ La fin d'activation d'un objet ne correspond pas à la fin de sa vie.
- ❑ Un même objet peut être activé plusieurs fois au cours de son existence.

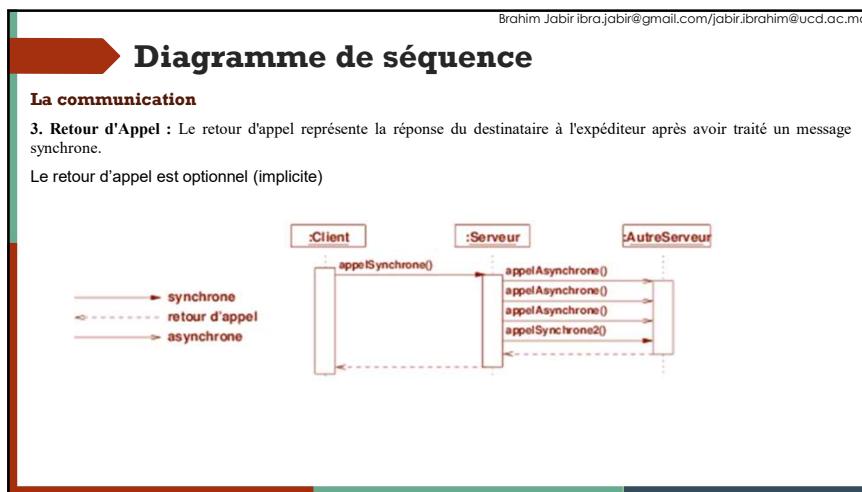
204



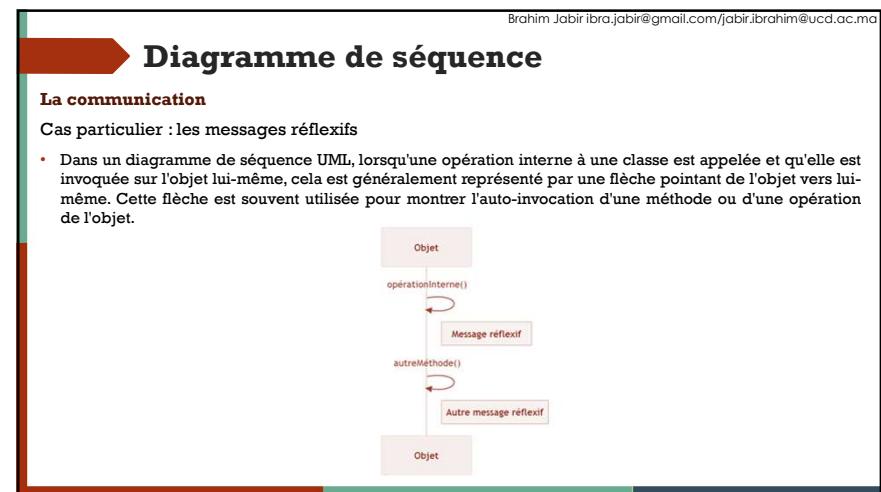
205



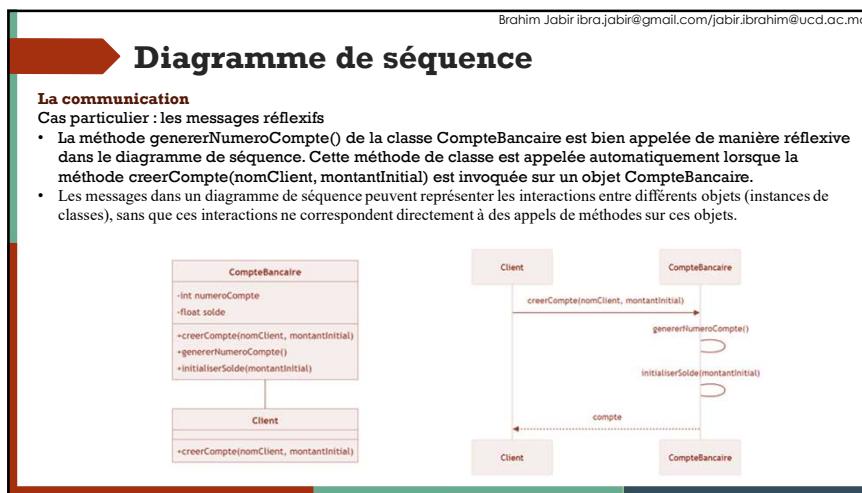
206



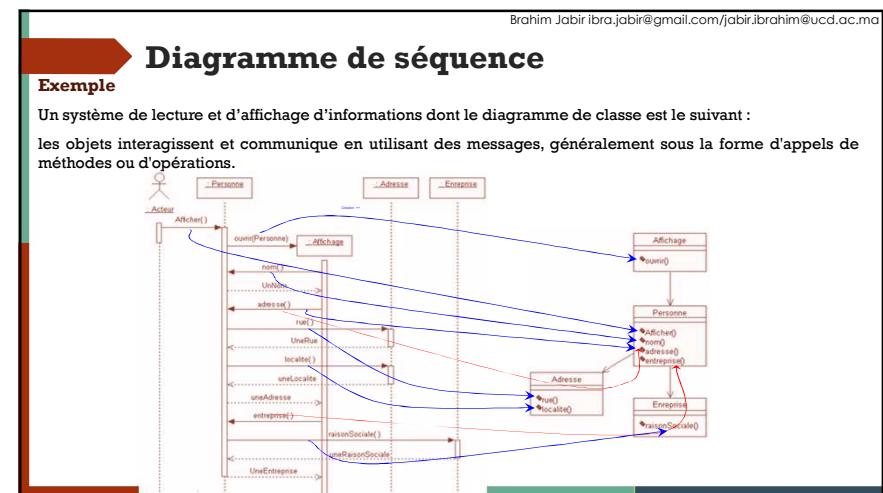
207



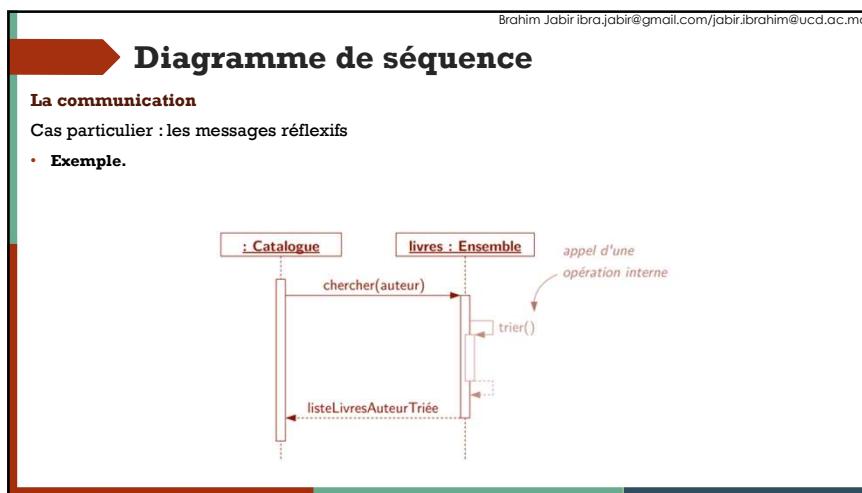
208



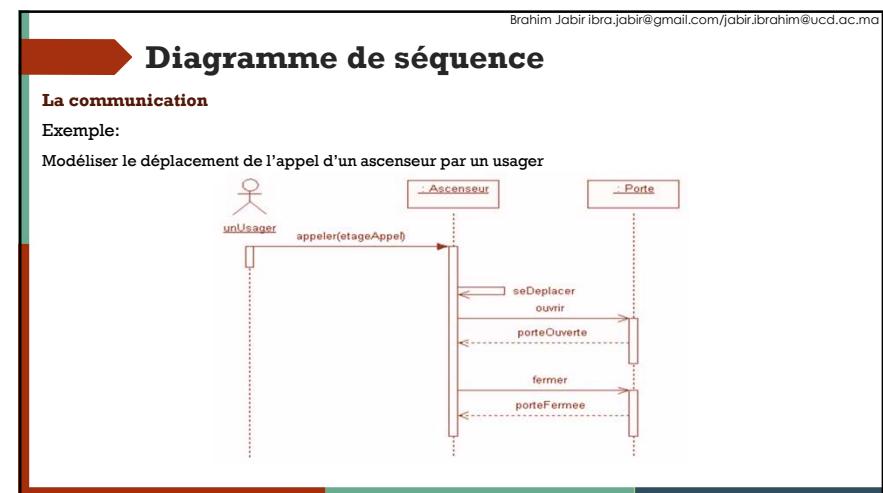
209



210



211



212

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Diagramme de séquence

La notion de contrôle

Deux Types de contrôle :

- Le contrôle centralisé
- Le contrôle décentralisé

Dans un modèle de contrôle centralisé, un seul objet détient le contrôle de la séquence d'exécution.

- Exemple: Un contrôleur d'application qui initie toutes les requêtes à d'autres composants et attend les réponses avant de continuer.

Dans un modèle de contrôle décentralisé, le contrôle de la séquence est distribué entre plusieurs objets participant à la coopération. Les objets s'échangent des messages et prennent des décisions de manière autonome, permettant ainsi une plus grande flexibilité.

Exemple:

- Un système de gestion de commandes où chaque module (comme le traitement de commandes, la facturation et l'expédition) communique avec les autres sans qu'un seul module ne contrôle l'ensemble du processus.

213

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Diagramme de séquence

La notion de contrôle

Deux Types de contrôle :

- Le contrôle centralisé
- Le contrôle décentralisé

Un seul objet détient le contrôle de la séquence : c'est le chef d'orchestre

Le contrôle de la séquence est dispatché entre les objets participant à la coopération : le contrôle se transmis par délégation

214

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Diagramme de séquence

Création et destruction d'objet

Création:

La création d'un objet est représentée en faisant pointer le message de création sur le rectangle qui symbolise l'objet créé.

retour obligatoire de l'instance créée

appel du constructeur de la classe

Session()

nouvelleSession : Session

exécution du constructeur

destroy()

destruction de l'objet (pas nécessairement à la suite d'un message)

215

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Diagramme de séquence

Création et destruction d'objet

Destruction:

- On peut détruire un objet pour le rendre explicite.
- La destruction d'un objet est représentée par un X à la fin de sa ligne de vie.
- Si l'objet est détruit par un autre objet (non par lui-même), le message issu de l'objet destructeur pointe sur le X ou juste à la suite de l'extrémité du message.
- Il est possible d'un objet se détruire lui-même.

« **Destroy** » et « **Create** » sont deux stéréotypes de messages.

Objet expéditeur du message 1

Objet destinataire du message 1 et expéditeur du message 2

Message de création

Message de destruction

DB(1)

DB(2)

DB(3)

DB(4)

1. Un message

2. Un autre message

3. <>create<>

4. Un dernier message

5. <>destroy<>

216

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Diagramme de séquence

Création et destruction d'objet (Voir la partie : les éléments de la COO)

Création d'un objet en Java :

Le message de création est représenté par le constructeur de la classe de l'objet créé.

Voiture Unevoiture= new Voiture()

217

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Diagramme de séquence

La communication

Étiquette des messages

- Une étiquette décrit les messages auxquels elle est attachée.
- Syntaxe générale :

```
[garde] *[itération]numéro_séquence:variable_retour := nom_opération(liste_arguments)
```

- nom_opération : le nom de l'opération à invoquer. (Si on veut juste appeler une méthode :nom_operation())
- liste_arguments : facultative, liste des arguments transmis à l'opération invoquée. Les éléments sont séparés par des virgules.
- variable_retour : facultative, valeur renvoyée par l'opération invoquée.
- garde : facultative, la condition préalable à l'envoi du message.
- Itération : facultative, le nombre d'envois du message ou conditions d'itération (* et les [] sont omis si pas d'itération).
- Numéro_séquence : facultatif, l'ordre du message dans la séquence (les : sont omis si Numéro_séquence n'est pas spécifié).

Utiliser uniquement les éléments nécessaires pour une syntaxe concise

218

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Diagramme de séquence

Structures de contrôle

Les messages conditionnels

- Dans les diagrammes de séquences, il est possible de modéliser les envois de messages conditionnels (structures de contrôles conditionnelles).
- Deux techniques de représentation :
 - Par pseudo-code (if X else end if).
 - Par garde ([X]) placée devant le message.
- Exemple 1/2 : Envio conditionnel de messages impliquant deux destinataires.**
- Exemple 3 : Envio conditionnel de messages impliquant un seul destinataire.**

Représentation par pseudo-code

Représentation par garde avec deux destinataires

Représentation par garde avec un seul destinataire

219

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Diagramme de séquence

Structures de contrôle

Les messages conditionnels

condition

a : Ascenseur

p : Portes

c : Cabine

[demandé = courant] ouvrir()

si la condition est vérifiée

[demandé ≠ courant] déplacer(étage)

sinon

220

Diagramme de séquence

Structures de contrôle

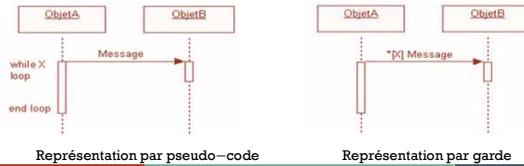
Les boucles

- Dans les diagrammes de séquences, il est possible de modéliser les envois de messages répétés (structures de contrôles itératives).

Deux techniques de représentation :

- Par pseudo-code (while X loop end loop) (X est la condition d'itération) ou par un langage quelconque comme java, etc. (UML ne précise aucun format pour les conditions).
- Par garde (*[X]) placé devant le message (* représente l'itération).

Exemple : ObjetA envoie un message à ObjetB tant que la condition X est vérifiée.



221

Diagramme de séquence

Structures de contrôle

Les boucles

Exemples de conditions d'itérations :

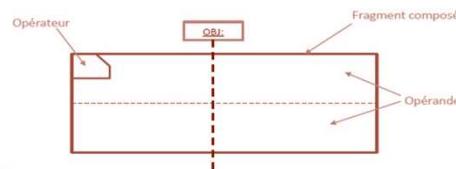
*[i:=1..10] message	Le message est envoyé 10 fois
*[x<10] message	Le message est envoyé de façon répétée jusqu'à ce que x soit plus grand ou égal à 10.
*[item not found] message	Le message est envoyé de façon répétée jusqu'à ce que l'item soit trouvé.
*[pour chaque employé] message	Le message est envoyé jusqu'à ce que tous les employés soient traités.
* message	La condition d'itération est omise (non spécifiée).

222

Diagramme de séquence

Fragments composés

- Les fragments composés représentent les expressions spécifiques dans la séquence.
- Les fragments composés doivent couvrir au moins une ligne de vie à tout moment, afin d'avoir une signification.
- Il est représenté par un rectangle dont le coin supérieur gauche contient un pentagone.
- Dans le pentagone figure le type du fragment : appelé opérateur d'interaction.
- La signification du fragment composé dépend fortement de l'opérateur d'interaction utilisé.

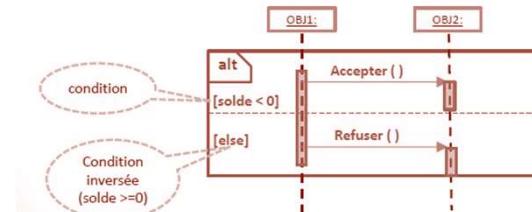


223

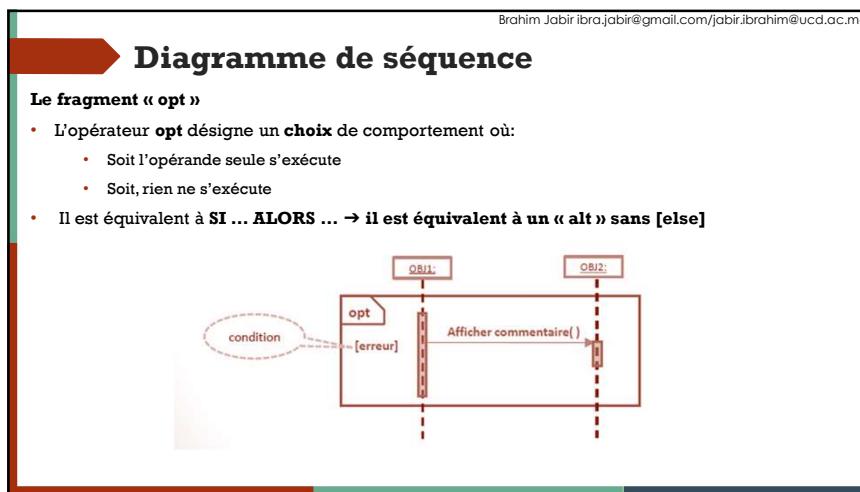
Diagramme de séquence

Le fragment « alt »

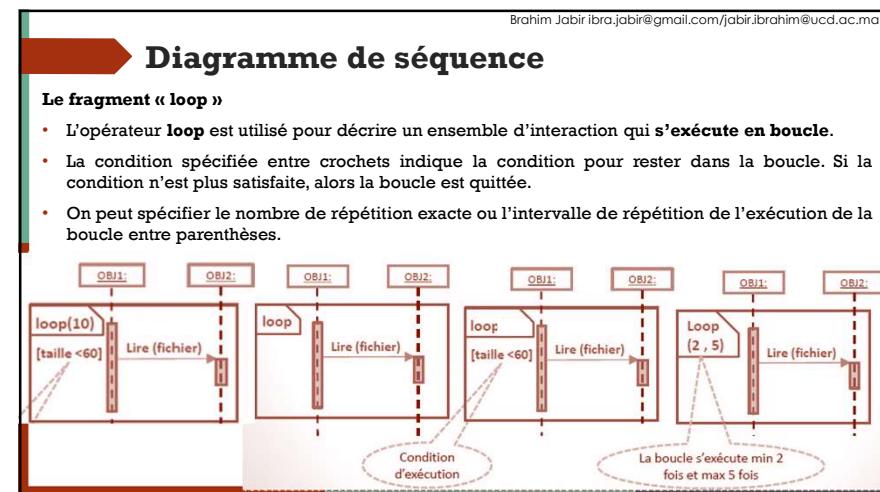
- L'opérateur alt désigne un choix ou une alternative: équivalent à SI ... ALORS ... SINON ...
- L'utilisation de l'opérateur else permet d'indiquer que la branche est exécutée si la condition du alt est fausse.
- Une seule des deux branches sera réalisée dans un scénario donné.



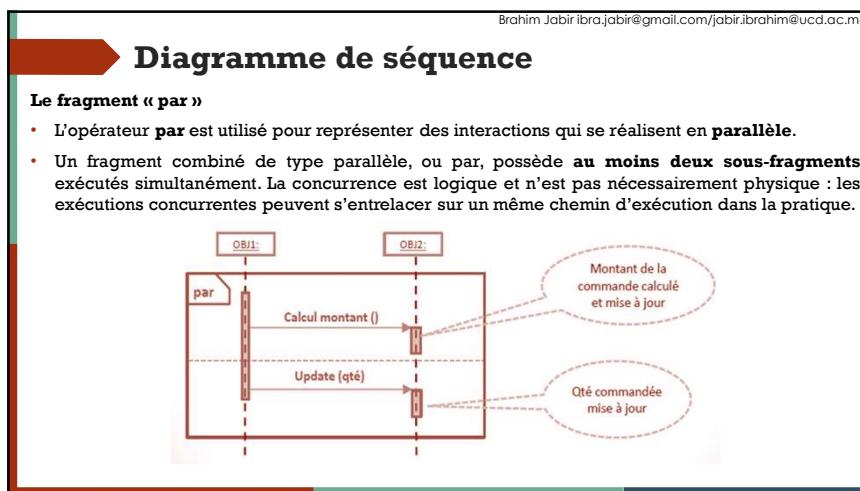
224



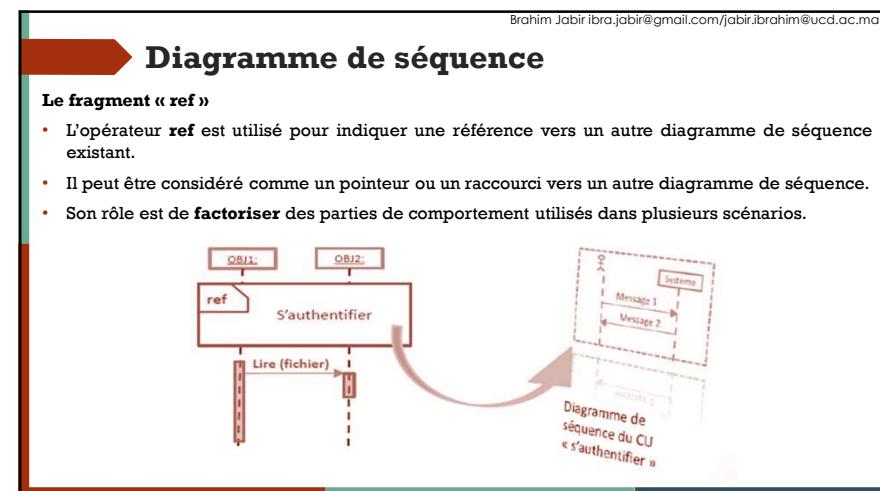
225



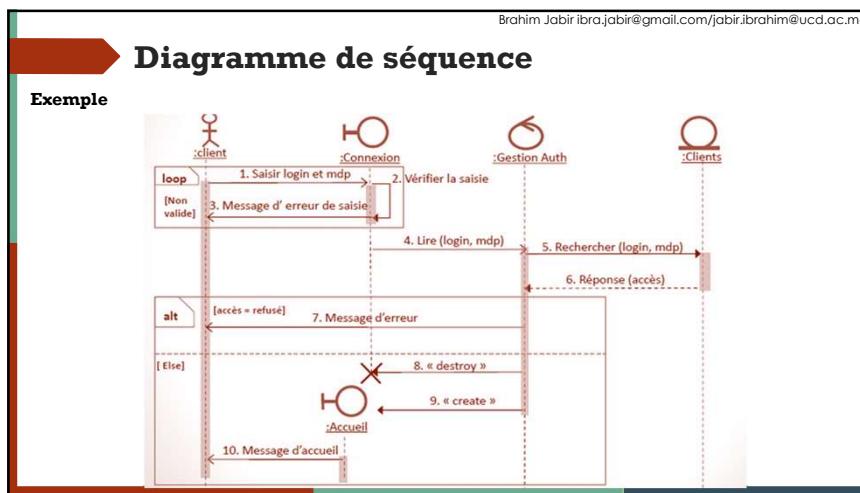
226



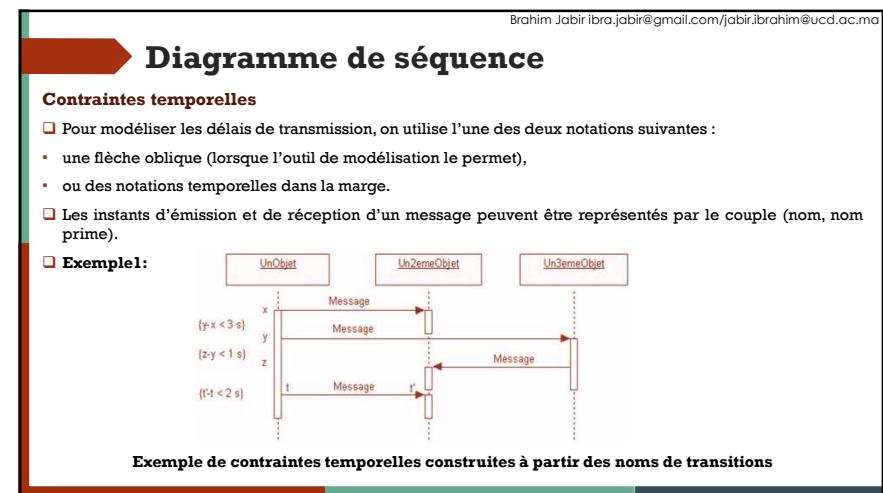
227



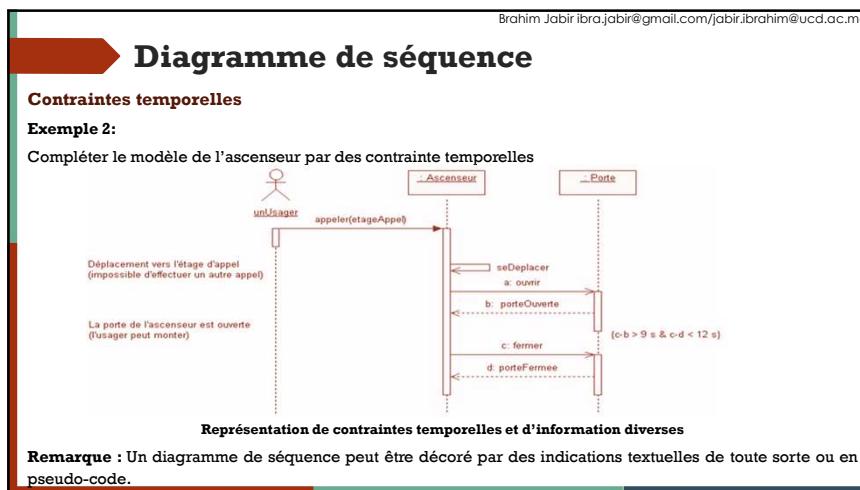
228



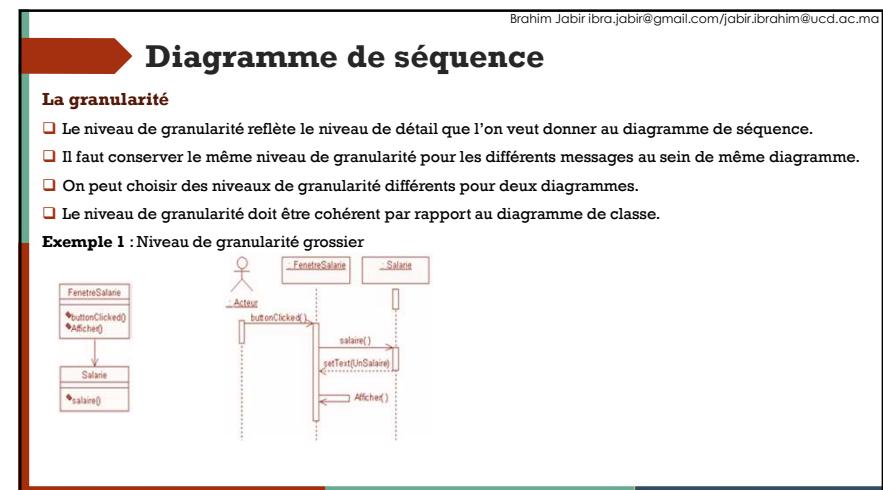
229



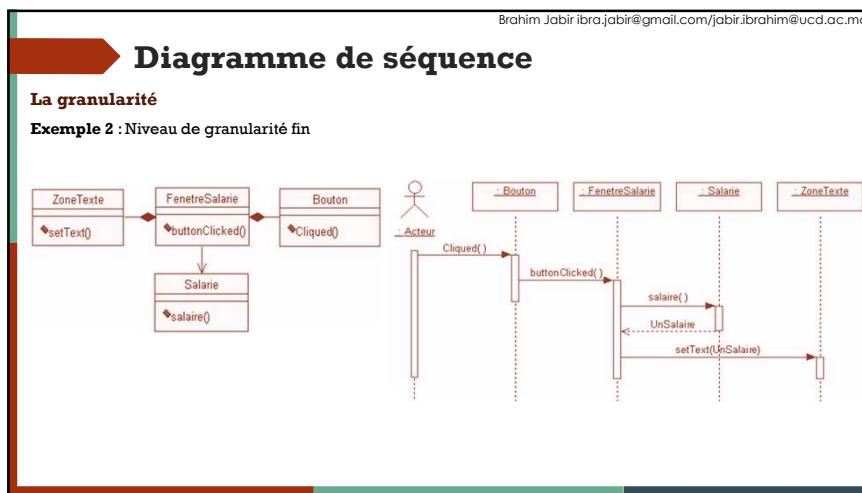
230



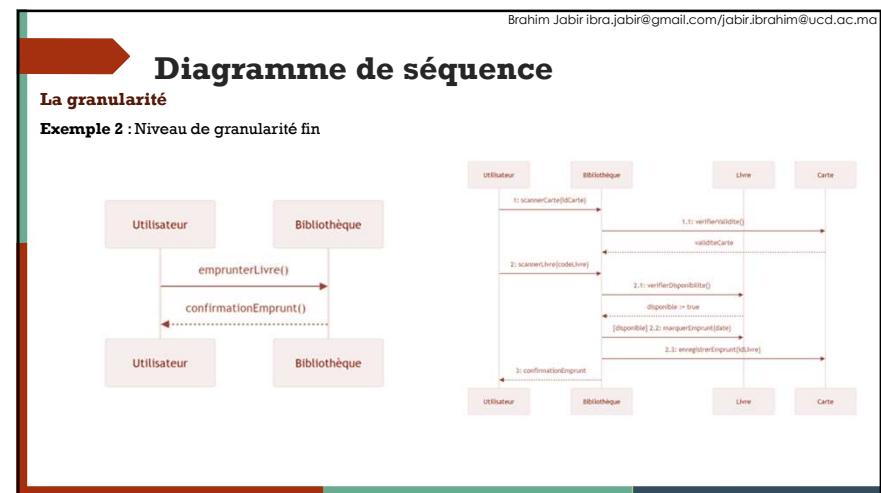
231



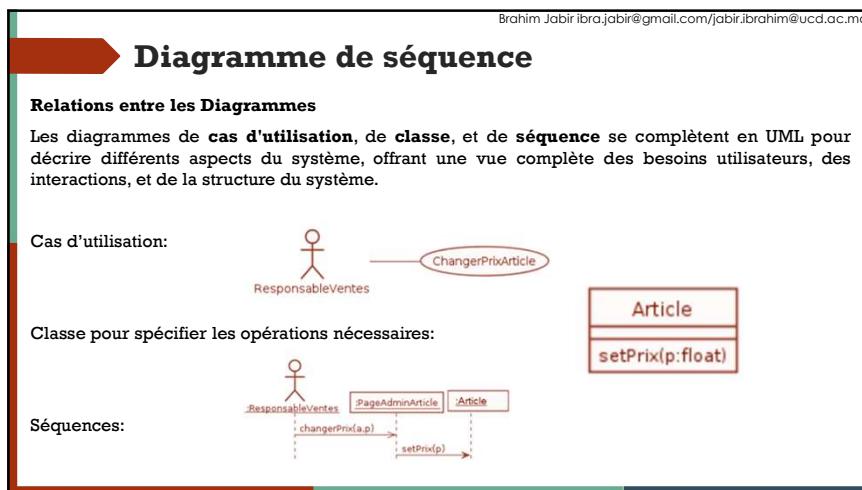
232



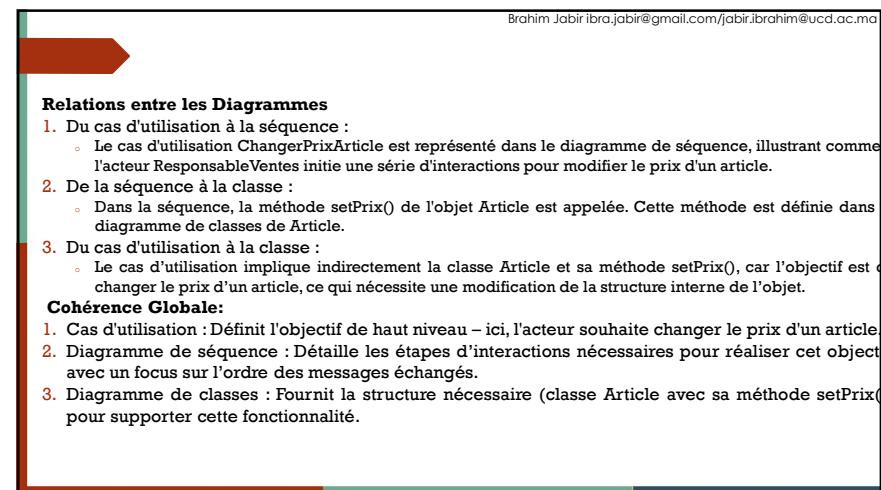
233



234



235



236

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Approches de Conception en UML

Il existe trois approches principales pour construire un modèle en UML : **descendante** (top-down), **ascendante** (bottom-up), et **itératrice**. Chaque approche a ses avantages en fonction de la situation et du niveau de connaissance initial du système.

Approche Descendante (Top-Down)

- Étapes :**
 - Commencez par le **diagramme de cas d'utilisation**, qui capture les besoins des utilisateurs.
 - Passez ensuite au **diagramme de séquence**, pour identifier les interactions nécessaires pour réaliser ces cas d'utilisation.
 - Enfin, raffinez le **diagramme de classes** pour y inclure les opérations requises.
- Avantages :**
 - Commence par les **besoins utilisateurs**, garantissant que le système répond à leurs exigences.
 - Permet d'**identifier les interactions clés** avant de s'intéresser aux détails techniques.
 - Déduit les **opérations nécessaires** à partir des interactions, assurant une conception orientée vers les usages.

237

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Approche Ascendante (Bottom-Up)

- Étapes :**
 - Définissez d'abord la **structure des classes**, en partant des entités de base du système.
 - Ajoutez ensuite les **opérations aux classes**.
 - Créez les **diagrammes de séquence** pour visualiser les interactions entre ces classes.
 - Vérifiez la cohérence avec les **cas d'utilisation**.
- Avantages :**
 - Utile si vous avez déjà une **bonne idée de la structure du système** ou de son implémentation.
 - Permet d'**élaborer les interactions à partir des classes**, favorisant une compréhension détaillée du système dès le départ.

238

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Approche Itérative (Recommandée)

- Étapes :**
 - Commencez par une **ébauche du diagramme de classe**, en incluant les attributs principaux.
 - Créez les **diagrammes de séquence** pour modéliser les interactions entre les objets.
 - Raffinez le **diagramme de classes** en ajoutant les opérations identifiées à partir des séquences.
 - Itérez entre les diagrammes pour garantir la **cohérence** et affiner le modèle au fil des découvertes.
- Avantages :**
 - Permet une **évolution progressive** du modèle, en ajustant les diagrammes au fur et à mesure des découvertes.
 - Assure une **meilleure cohérence** entre les diagrammes, car chaque aspect du système est révisé et affiné plusieurs fois.
 - Facilite la **découverte de nouvelles opérations** ou classes nécessaires, par l'ajustement continu.

239

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Diagramme de séquence

Exercice d'application

Le déroulement de réservation de vol est le suivant:

- Le Client sélectionne un vol.
- Une validation de nombre de place automatique du système se déclenche.
- Si le nombre de places réservées est inférieur au nombre de places disponibles dans un vol, la création de la réservation se lance, sinon une erreur s'affichera au client.

240

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Diagramme de séquence

Exercice d'application

Le déroulement de création de vol est le suivant:

- Le gestionnaire sélectionne un avion.
- Ensuite il doit sélectionner un aéroport de départ et un autre d'arrivée
- Puis il crée un vol après avoir spécifier les autres informations requis.

241

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Diagramme d'activité

Diagramme d'activité est utilisé pour:

- Modéliser un workflow dans un use case ou entre plusieurs use cases.
- Les diagrammes d'activité affichent le flux de travail d'un point de départ à un point d'arrivée en détaillant les nombreux chemins de décision existant dans la progression des événements contenus dans l'activité.
- Spécifier une opération (décrire la logique d'une opération) :la logique métier

Le diagramme d'activité est le plus approprié pour modéliser la dynamique d'une tâche, d'un use case.

242

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Diagramme d'activité

Diagramme d'activité =

- Ensemble d'activités liés par:
 - *Transition (sequentielle)*
 - *Transitions alternatives (conditionnelle)*
 - *Synchronisation (disjonction et conjonctions d'activités)*
 - *Itération*
- + 2 états: état de *départ* et état de *terminaison*
- *Swimlanes*: représente le lieu, le responsable des activités.

243

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Diagramme d'activité

Un nœud d'activité

Un nœud d'activité est un type d'élément abstrait permettant de représenter les étapes le long du flot d'une activité. Il existe trois familles de nœuds d'activités :

- les nœuds d'exécutions (*executable node* en anglais) ;
- les nœuds objets (*object node* en anglais) ;
- les nœuds de contrôle (*control nodes* en anglais).

244

Brahim Jabir ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Diagramme d'activité

Noeuds de contrôle

- Le noeud représentant une action, qui est une variété de noeud exécutable.
- Un noeud objet.
- Un noeud de décision ou de fusion.
- Un noeud de bifurcation ou d'union.
- Un noeud initial.
- Un noeud final.
- Un noeud final de flux.

245

Brahim Jabir ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Notion du diagramme d'activité

Exemple

246

Brahim Jabir ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Diagramme d'activité

noeud initial et noeud final

Nœud initial:

Le nœud initial indique le point de départ du flux d'activité dans le diagramme.

Une activité peut avoir plusieurs nœuds initiaux, par exemple dans le cas d'une activité démarrée par différents événements.

Le nœud initial n'est pas une action en soi, il représente simplement le point de départ du flux.

Le noeud initial est représenté par un petit cercle noir.

Exemple:

Dans un système de billetterie automatique, il peut y avoir deux façons de démarrer une interaction : avec une carte de transport ou en achetant un billet.

- Nœud initial 1 : Utilisation de la carte de transport
- Début du flux : "Scanner la carte de transport"
- Nœud initial 2 : Achat sans carte
- Début du flux : "Sélectionner l'option achat sans carte"

247

Brahim Jabir ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Diagramme d'activité

noeud initial et noeud final

Nœud final: Un noeud final est un noeud de contrôle possédant un ou plusieurs arcs entrants. Il existe deux types de noeud finaux :

- Nœud de fin d'activité :**

Lorsque l'un des arcs d'un noeud de fin d'activité est activé, l'exécution de l'activité enveloppante s'achève et tout noeud ou flux actif au sein de l'activité enveloppante est abandonné.

Dans un système de GAB, si un utilisateur entre un code PIN incorrect trois fois, l'opération est annulée pour des raisons de sécurité.

- Nœud de fin de flux :**

Lorsqu'un flux d'exécution atteint un noeud de fin de flux, le flux en question est terminé, mais cette fin de flux n'a aucune incidence sur les autres flux actifs de l'activité enveloppante.

Dans un système de GAB, lorsqu'un ticket est introuvable, cela peut interrompre un sous-flux sans pour autant affecter le flux principal de récupération des billets.

248

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Diagramme d'activité

Un nœud d'action

- Un nœud d'action est un nœud d'activité exécutable qui constitue l'unité fondamentale de fonctionnalité exécutable dans une activité.
- L'exécution d'une action représente une transformation ou un traitement spécifique dans le système modélisé. Les actions sont généralement liées à des opérations qui sont directement invoquées.
- Graphiquement, un nœud d'action est représenté par un rectangle aux coins arrondis qui contient sa description textuelle. Cette description textuelle peut aller d'un simple nom à une suite d'actions réalisées par l'activité.
- Actions système** (exécutées automatiquement par le système
 - Validation automatique des données
 - Enregistrement en base de données
- Actions utilisateur** (nécessitent une intervention humaine)
 - Saisie de données
 - Validation manuelle

249

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Diagramme d'activité

Nœud exécutable: Notation spéciale

- Le nœud "Accept Event" est utilisé pour représenter le point où un processus ou une activité attend un événement spécifique pour se produire.(Point d'attente d'un événement externe, bloqué jusqu'à réception).
- Exemple : attente d'une réponse utilisateur ou d'une validation
- Nœud "Send Signal" utilisé pour déclencher des actions ou des transitions dans d'autres activités ou processus.(Émet un signal vers d'autres processus)
- Exemple : envoi d'une notification ou déclenchement d'une action parallèle
- Le nœud "Accept Time Event" est placé dans le diagramme d'activité pour indiquer que l'activité actuelle doit attendre pendant un certain laps de temps avant de passer à l'étape (Crée une pause temporelle) suivante.
- Exemple : délai d'attente de 24h avant de poursuivre

250

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Diagramme d'activité

Noeud exécutable: Notation spéciale

- Les nœuds "Accept Event"** :
 - "Déetecter arrivée du train"
 - "Déetecter passage du train"
 - Ces nœuds attendent un signal externe (capteurs)
 - Le processus est bloqué jusqu'à la détection
- Les nœuds "Send Signal"** :
 - "Faire clignoter les feux"
 - "Éteindre les feux"
 - Ces actions envoient des commandes aux équipements
 - L'exécution continue sans attendre de confirmation
- Le nœud "Time Event"** :
 - Attendre 3 secondes"
 - Délai de sécurité entre l'activation des feux et l'abaissement de la barrière
 - Permet aux usagers de la route de réagir

251

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Diagramme d'activité

Les activités (activity) :

Une activité définit un comportement décrit par un séquencement organisé d'unités dont les éléments simples sont les actions.

Une activité est représentée comme un rectangle à coins arrondis enfermant toutes les actions, les flux de contrôle et d'autres éléments qui composent l'activité.

252

Brahim Jabir ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Diagramme d'activité

Transition

- Le flux dans un diagramme d'activité UML est principalement représenté par des flèches appelées transitions.
- Le passage d'une activité vers une autre est matérialisé par une transition.
- Graphiquement les transitions sont représentées par des flèches en traits pleins qui connectent les activités entre elles
- Elles sont déclenchées dès que l'activité source est terminée et provoquent automatiquement et immédiatement le début de la prochaine activité à déclencher (l'activité cible).
- Contrairement aux activités, les transitions sont franchies de manière atomique, en principe sans durée perceptible.
- L'étape suivante ne peut pas commencer avant que l'étape actuelle soit terminée

```

graph LR
    A([Etablir commande]) --> B([Livrer Commande])
  
```

253

Brahim Jabir ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Diagramme d'activité

Nœud de décision (decision node)

Un nœud de décision est un nœud de contrôle qui permet de faire un choix entre plusieurs flots sortants. Il possède un arc entrant et plusieurs arcs sortants.

Ces derniers sont généralement accompagnés de conditions de garde pour conditionner le choix. Graphiquement, nous représentons un nœud de décision par un losange

```

graph LR
    Start(( )) --> Decision{Decision Node}
    Decision -- "[condition is true]" --> ActionTrue([Action on True])
    ActionTrue --> End(( ))
    Decision -- "[condition is false]" --> ActionFalse([Action of False])
    ActionFalse --> Merge(( ))
    Merge --> End
  
```

254

Brahim Jabir ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Diagramme d'activité

```

graph TD
    Start(( )) --> V[Verification Commande]
    V --> Decision{Commande Valid?}
    Decision -- Oui --> EC([Enregistrement Commande])
    EC --> End(( ))
    Decision -- Non --> RC([Rejet Commande])
    RC --> IE([Informer Erreur au Client])
    IE --> Merge(( ))
    Merge --> End
  
```

255

Brahim Jabir ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Diagramme d'activité

Nœud de fusion (merge node)

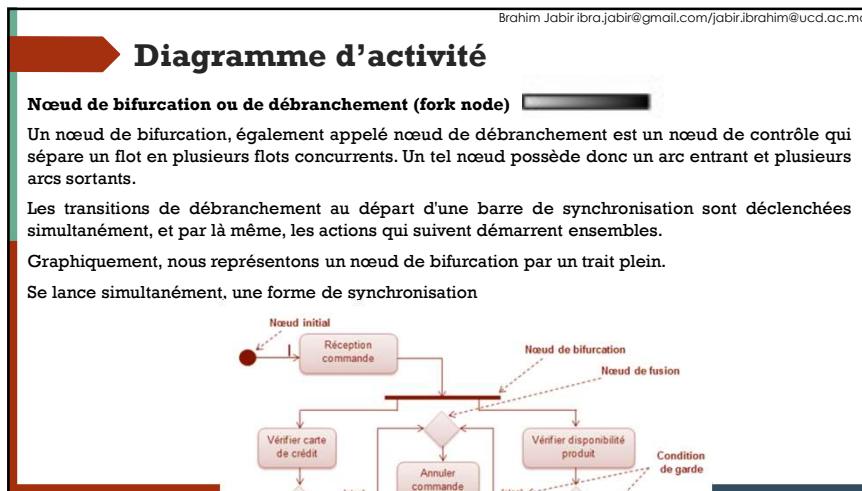
Un nœud de fusion est un nœud de contrôle qui rassemble plusieurs flots alternatifs entrants en un seul flot sortant.

Il est utilisé pour indiquer que plusieurs chemins divergents se rejoignent en un seul flux.

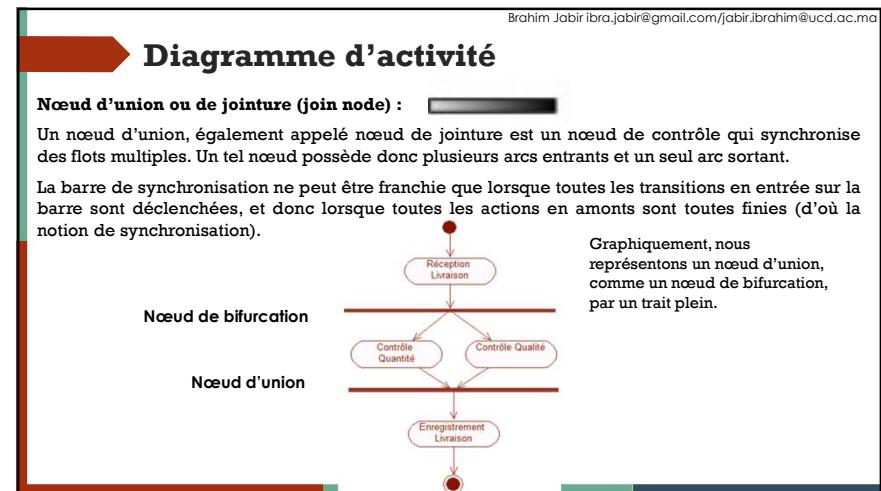
```

graph LR
    Start(( )) --> Decision{Decision Node}
    Decision -- "[condition is true]" --> ActionTrue([Action on True])
    ActionTrue --> Merge(( ))
    Decision -- "[condition is false]" --> ActionFalse([Action of False])
    ActionFalse --> Merge
    Merge --> End(( ))
  
```

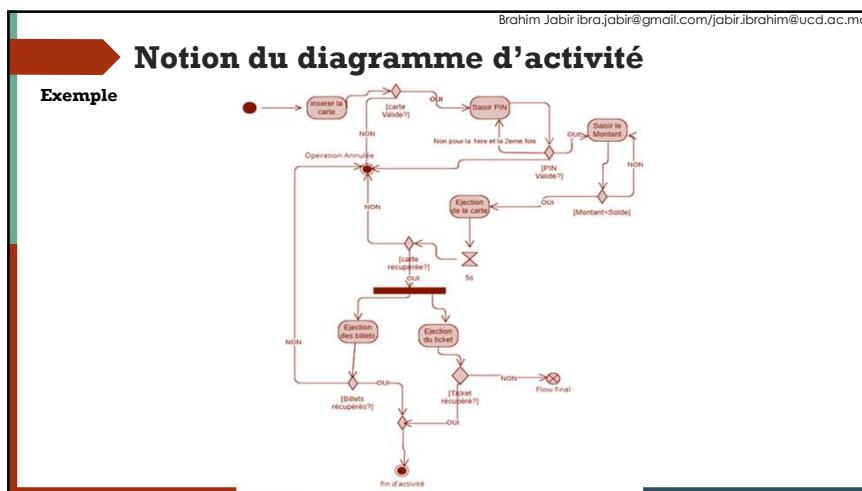
256



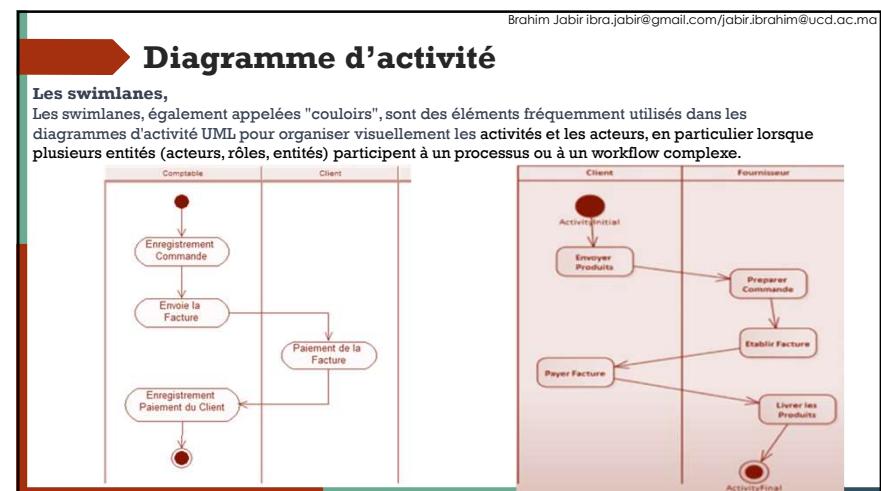
257



258



259



260

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Diagramme d'activité

Exercice 1: Cafetière

Construire un diagramme d'activité représentant l'utilisation d'une cafetière électrique:

- ▶ premier état: chercher du café
- ▶ dernier état: Servir du café

Les autres sont:

- 1.Mettre un filtre
- 2.Mettre du café
- 3.Remplir le réservoir d'eau
- 4.Allumer la cafetière
- 5.Préparer une tasse vide

261

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Diagramme d'activité

Exercice 2: Commander un produit

Construire un diagramme d'activité pour modéliser le processus de commander d'un produit. Le processus concerne les acteurs suivants:

- ▶ **Client:** qui commande un produit et qui paie la facture
- ▶ **Responsable Vente:** qui s'occupe de traiter et de facturer la commande du client
- ▶ **Responsable Entrepôt:** qui est responsable de sortir les articles et d'expédier la commande.
- ▶ **Responsable Caisse:** qui encaisse l'argent du client

262

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Diagramme d'activité

Exercice d'application

Créer un diagramme d'activité pour le « use case » 'Créer une fiche de réparation':

- Pour créer une fiche de réparation, le chef d'atelier saisit les critères de recherche de voitures dans le système. Le logiciel de gestion des réparations lui donne la liste des voitures correspondant aux critères entrés. Si la voiture existe, le chef d'atelier va sélectionner la voiture. Le logiciel va, ensuite, fournir les informations sur le véhicule. Si la voiture est sous garantie, le chef devra saisir la date de demande de réparation. Si la voiture n'existe pas, le chef va saisir les informations concernant ce nouveau véhicule. Dans tous les cas, le chef d'atelier devra saisir la date de réception et de restitution. Si le dommage de la voiture est payé par l'assurance, le logiciel va fournir une liste d'assurances au chef d'atelier. Ce dernier sélectionnera l'assurance adéquate. Enfin, le logiciel enregistre la fiche de réparation.

263

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Diagramme d'états-transitions

Rappel

- Diagramme de classes
 - Montre la structure statique et globale de système.
 - La visualisation des classes et des relations entre elles.
- Diagramme de cas d'utilisation
 - Montre les relations possibles entre les différents acteurs
 - Montre les relations possibles entre les services du système
- Diagramme de séquence
 - Interactions entre les objets de l'application avec les acteurs

264

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Diagramme d'états-transitions

Rappel

- Diagramme de classes
 - Montre la structure statique et globale de système.
 - La visualisation des classes et des relations entre elles.
- Diagramme de cas d'utilisation
 - Montre les relations possibles entre les différents acteurs
 - Montre les relations possibles entre les services du système
- Diagramme de séquence
 - Interactions entre les objets de l'application avec les acteurs

265

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Diagramme d'état de transition

Objectif : Décrire le comportement dynamique d'une entité (logiciel, composant, objet...)

Comportement décrit par états + transitions entre les états

- État : abstraction d'un moment de la vie d'une entité pendant lequel elle satisfait un ensemble de conditions
- Transition : changement d'état.
- Les événements auxquelles ils réagissent.

266

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Diagramme d'états-transitions

C'est un graphe composé de:

- Un ensemble de nœuds(états du système)
- Un ensemble d'arc (transitions entre les états)
- Un diagramme d'état de transition est représenté par un automate

267

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Diagramme d'états-transitions

Le diagramme état-transition manipule plusieurs concepts :

- État
- Transition
- Événement
- Garde
- ...

268

Diagramme d'états-transitions

- Chaque objet est à un moment donné dans un état particulier. Les états se représentent sous la forme de rectangles arrondis ;
- Chaque état possède un nom qui l'identifie.
- Un état est toujours l'image de la conjonction instantanée :
 - des valeurs contenues par les attributs de l'objet,
 - et de la présence ou non de liens, de l'objet considéré vers d'autres objets.

269

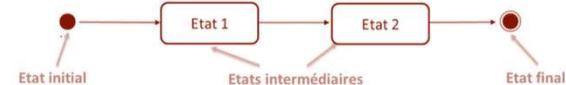
Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Notion

État : abstraction d'un moment de la vie d'un objet, pendant lequel il satisfait un ensemble de conditions.

Types d'états:

- Etat initial: Initialisation du système, exécution du constructeur de l'objet
- Etat final: Fin de vie du système, destruction de l'objet
- Etat intermédiaire: Etapes de la vie du système, de l'objet



270

Diagramme d'états-transitions

État

- Un automate à états finis est graphiquement représenté par un graphe comportant des états, matérialisés par des rectangles aux coins arrondis, et des transitions, matérialisées par des arcs orientés liant les états entre eux.

état simple

Le nom de l'état peut être spécifié dans le rectangle et doit être unique dans le diagramme d'états-transitions, ou dans l'état enveloppant. On peut l'omettre, ce qui produit un état anonyme.

271

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Diagramme d'états-transitions

État

- Le diagramme de classes suivant représente des personnes qui travaillent pour des sociétés.



- Les personnes ne possèdent pas toutes un emploi et se trouvent, à un moment donné, dans l'un des états suivants :
 - en activité, au chômage ou à la retraite.
- La figure suivante visualise les trois états d'une personne.



272

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

État

- Pour connaître la situation d'une personne en particulier, il faut étudier la conjonction suivante :
 - âge de la personne,
 - présence d'un lien vers une société.

```

graph LR
    Societe[":Société"] --- p1["p1:Personne  
age=30"]
    Societe --- p2["p2:Personne  
age=45"]
    Societe --- p3["p3:Personne  
age=75"]
    p1 --- AuChomage[Au chômage]
    p1 --- EnActivite[En activité]
    p1 --- ALaRetraite[À la retraite]
  
```

273

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Diagramme d'états-transitions

État

- La figure suivante montre un exemple simple d'automate à états finis.
- Cet automate possède deux états (*Allumé* et *Eteint*) et deux transitions correspondant au même événement : la pression sur un bouton d'éclairage domestique.
- Cet automate à états finis illustre en fait le fonctionnement d'un tél rupteur (bouton poussoir) dans une maison. Lorsque l'on appuie sur un bouton d'éclairage, la réaction de l'éclairage associé dépendra de son état courant (de son historique) : s'il la lumière est allumée, elle s'éteindra, si elle est éteinte, elle s'allumera.

```

graph TD
    Allume[Allumé] -- pression --> Eteint[Eteint]
    Eteint -- pression --> Allume
  
```

274

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Diagramme d'états-transitions

Intérêt :

- Vue synthétique de la dynamique de l'entité
- Regroupe un ensemble de scénarios

```

graph TD
    Ouverte[Ouverte] -- ouvrir --> Fermee[Fermée]
    Fermee -- fermer --> Ouverte
    Fermee -- déverrouiller --> Verrouillee[Verrouillée]
    Verrouillee -- verrouiller --> Fermee
  
```

275

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Notion:

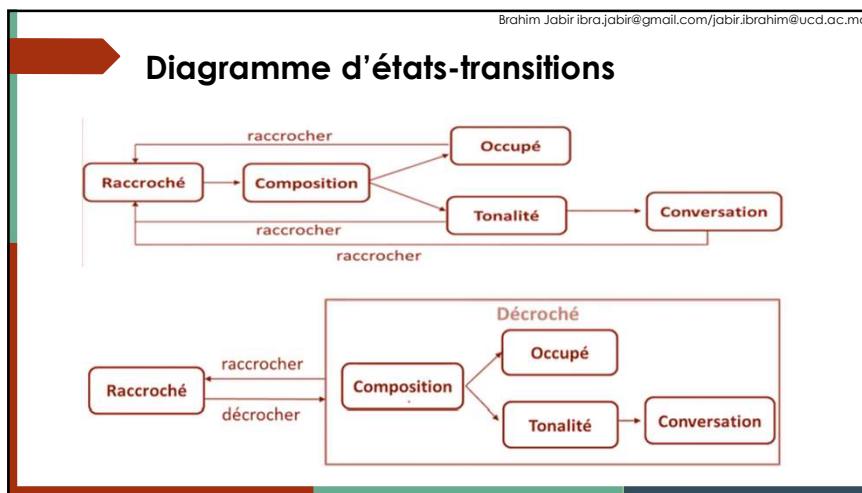
Un état Composite: un état qui contient des sous états

On définit des sous-états et des super-états

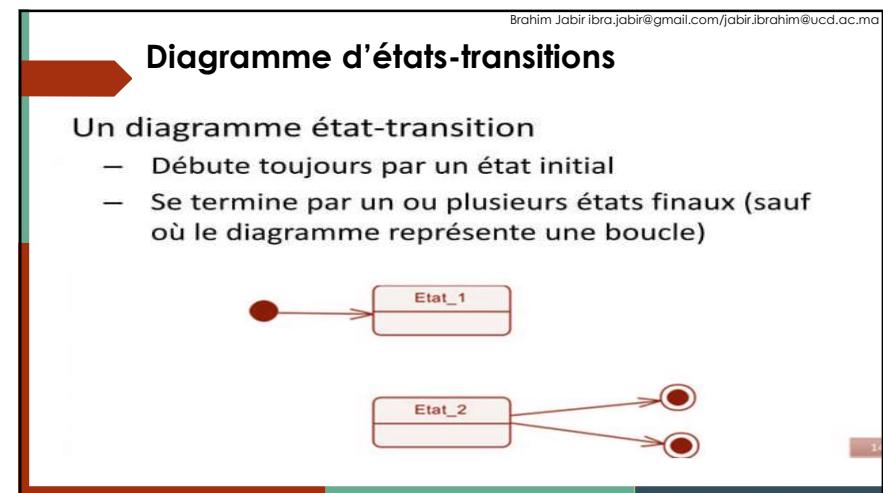
Objectifs :

- Hiérarchiser les états
- Structurer les comportements complexes
- Factoriser les actions

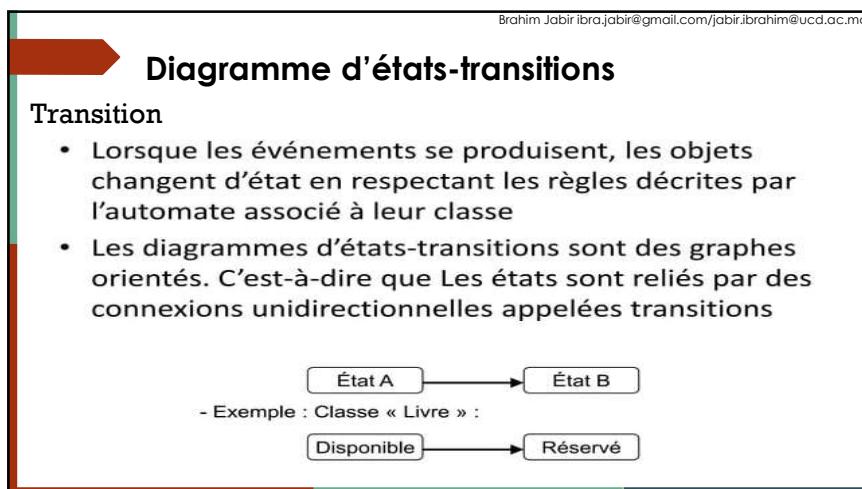
276



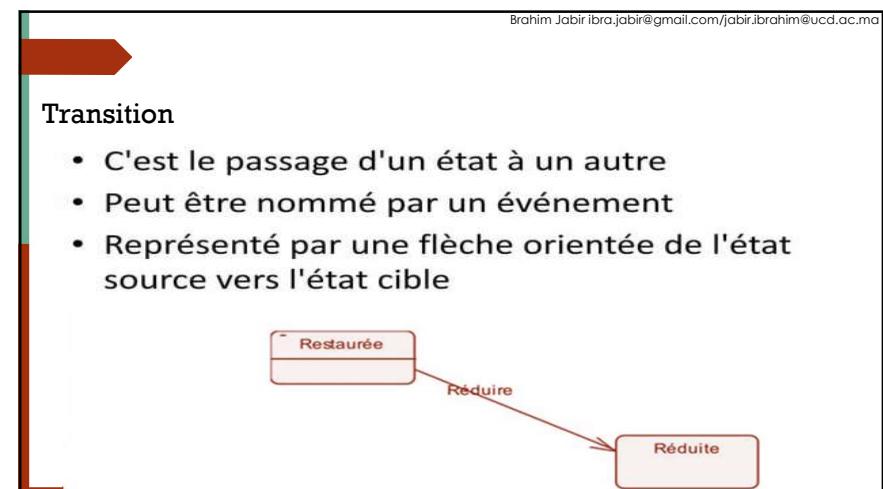
277



278



279



280

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Diagramme d'états-transitions

Événement

Les événements sont généralement des déclencheurs interne ou externe, qui provoquent un changement d'état.

Types d'événements :

- Signal: réception d'un message asynchrone
- Appel d'une opération (synchrone): liée aux cas d'utilisation, opération du diagramme de classes...
- Satisfaction d'une condition booléenne: when(cond), évaluée continuellement jusqu'à ce qu'elle soit vraie
- Temps
 - Date relative: when(date = date)
 - Date absolue: after(durée)

```

graph LR
    A[Etat A] -- "Evt" --> B[Etat B]
  
```

281

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Evènement:

- Fait (externe) survenu qui déclenche une transition (changement d'états)
- Peut être réflexif et conduire au même état
- Conduit à l'appel d'une méthode de la classe de l'objet
- Peut posséder des attributs :
 - paramètres portés par des événements
 - Représentés entre parenthèses

282

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Diagramme d'états-transitions

Évènement:

- Syntaxe d'un événement :
 - Nom_De_L_Evénement (Nom_De_Paramètre : Type, ...)
- La description complète d'un événement t est donnée par :
 - nom de l'événement
 - liste des paramètres
 - objet expéditeur
 - objet destinataire
 - sa description textuelle

```

graph TD
    A[En activité] -- "embbauche" --> B[Au chômage]
    B -- "Plus de 60 ans" --> C[À la retraite]
    C -- "Perte d'emploi" --> B
  
```

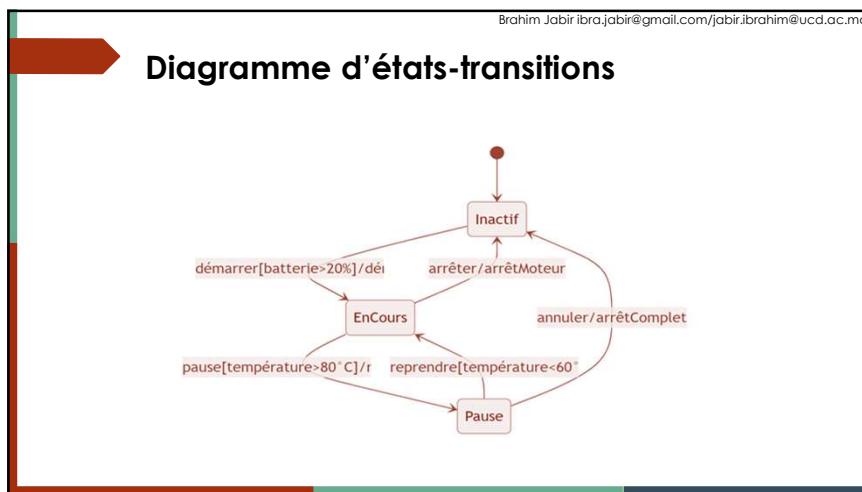
283

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

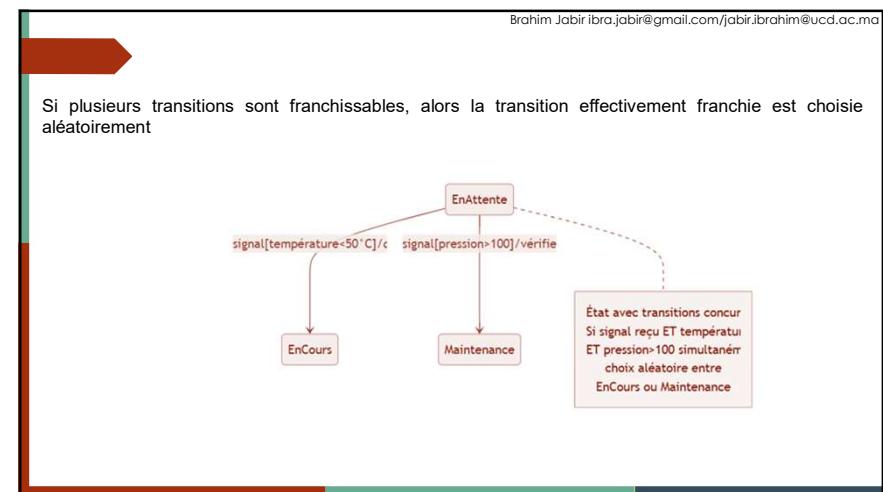
Événement, condition et action d'une transition

- Syntaxe complète
 - événement [condition]/action
- La transition est déclenchée par un événement
- La transition est effectivement franchie si la condition (sur l'état de l'objet) est valide
- Le franchissement déclenche l'exécution de l'action de la transition
 - Cette action est considérée comme immédiate et atomique

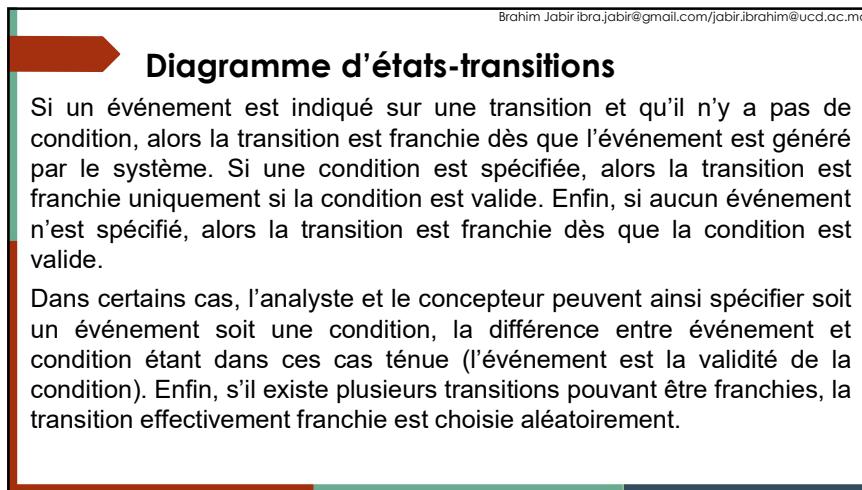
284



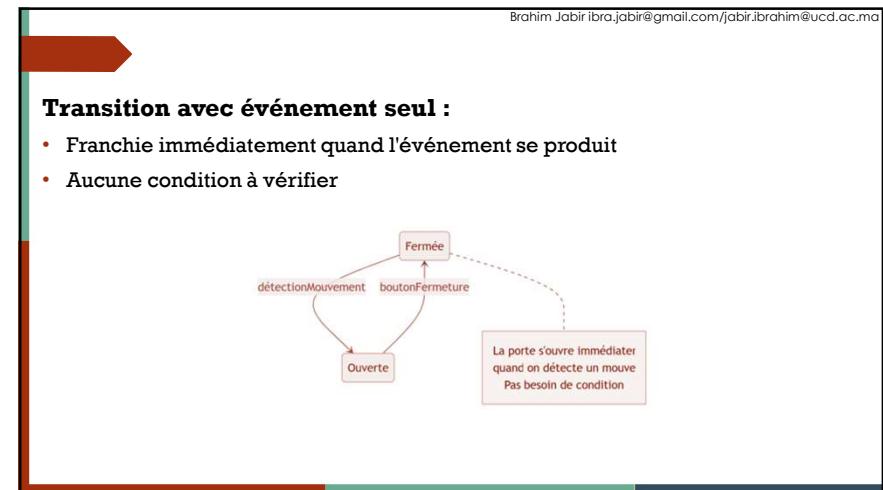
285



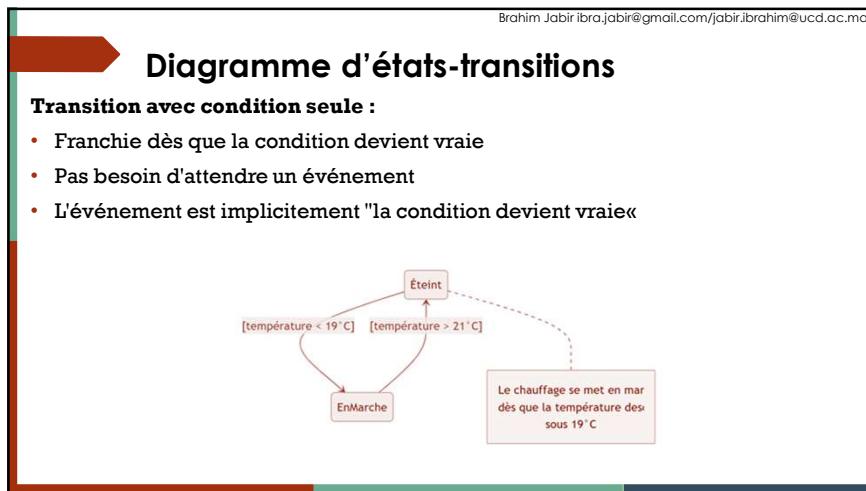
286



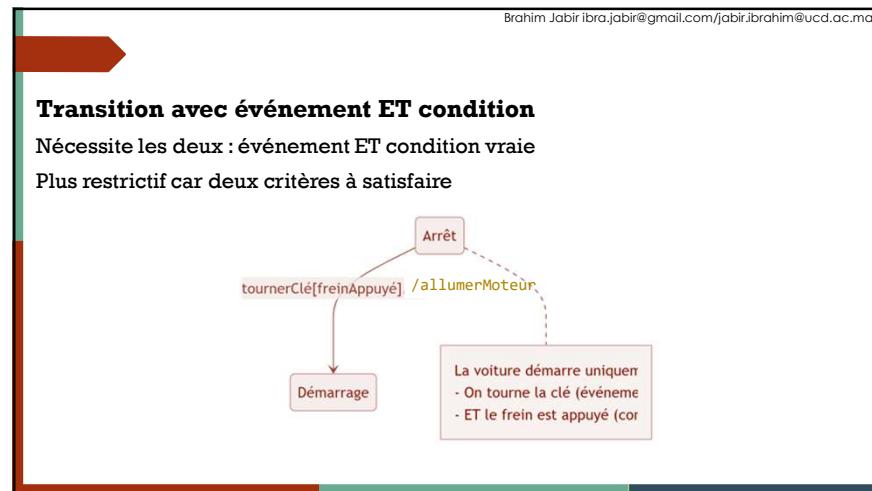
287



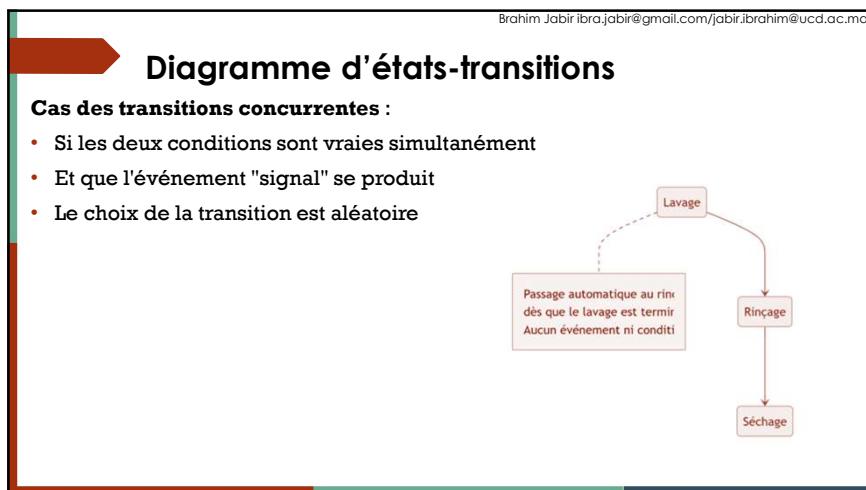
288



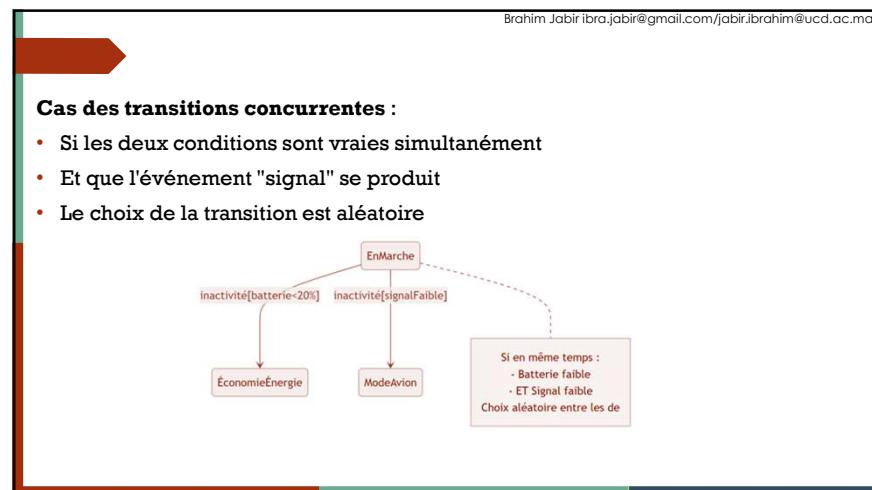
289



290



291



292

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Diagramme d'états-transitions

Les événements internes d'état

Les événements internes d'état sont des actions qui :

1. Se produisent à l'intérieur d'un état
2. Ne provoquent PAS de changement d'état (pas de transition vers un autre état)
3. Permettent de réagir à des événements tout en restant dans le même état

```

    graph TD
      Etat[Etat] --> Entry[Entry [condition] / action]
      Entry --> Do[Do / activité]
      Do --> Event1[Event 1 [condition] / action]
      Event1 --> Event2[Event 2 [condition] / action]
      Event2 --> ...[...]
      ... --> Exit[Exit [condition] / action]
      Exit --> Etat
  
```

293

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

entry/ [action]

Définition : Action exécutée une seule fois à l'entrée de l'état

Syntaxe : entry/ nomAction()

Exemples:

- entry/ initialiserSystème()
- entry/ éteindreLed()

294

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Diagramme d'états-transitions

do/ [action]

Définition : Action exécutée en continu pendant l'état
(Exécutées EN CONTINU tant qu'on reste dans l'état)

Syntaxe : do/ nomAction()

Exemples:

- do/ surveillerTempérature()
- do/ tournerPlateau()

295

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Diagramme d'états-transitions

[nomEvénement]/ [action]

Définition : Action exécutée en réponse à un événement spécifique ou Déclenchés par des conditions spécifiques pendant l'état.

Syntaxe : nomEvénement/ nomAction()

Exemples:

- alarme/ activerSirène()
- event1/ ajusterTempérature()
- event2/ vérifierPression()

296

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Diagramme d'états-transitions

exit/ - Actions de sortie :

Exécutées UNE SEULE FOIS au moment de quitter l'état

exit : dernière action avant de quitter

Exemples:

- exit/ sauvegarderDonnées()
- exit/ fermerConnexions()

297

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Diagramme d'états-transitions

Ordre d'exécution :

1. **entry** : première action exécutée
2. **do** : exécution continue
3. **events** : en réponse à des triggers
4. **exit** : dernière action avant de quitter

EnProduction

 entry/ démarrerSystème()

 do/ produirePièces()

 exit/ arrêterProduction()

298

Brahim.Jabir.ibra.jabir@gmail.com/jabir.ibrahim@ucd.ac.ma

Diagramme d'états-transitions

Exercice:

Elaborer le diagramme d'état de transition de l'objet réservation de de vol:

299