

lab-sticc.univ-brest.fr/~babau/

Diagrammes de classe UML

Jean-Philippe Babau

Département Informatique, UFR Sciences, UBO
Laboratoire Lab-STICC



UBO

Plan

- " Introduction aux diagrammes de classe
- " Description des diagrammes de classe
- " « Bien » faire des diagrammes de classe

jean-philippe.babau@univ-brest.fr 3

UBO

Besoins

- " Représenter
 - . Des concepts : les classes
 - " Un nom
 - . Des propriétés simples pour chaque concept : les attributs
 - " Un nom
 - " Un type
 - . Des relations entre concepts : les associations
 - " Un ou deux noms
 - " Multiplicité
 - . Des héritages : les relations d'héritage entre classes
 - . Des opérations : les services de la classe

jean-philippe.babau@univ-brest.fr 4

Exemple : code Java

UBO

```

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

public class Reseau {
    private Gare lesGares[];
    private List<Troncon> lesTroncons = new ArrayList<Troncon>();
}

public class Gare {
    private String nom;
}

public abstract class Troncon {
    protected Troncon estRelieA[] = new Troncon[3];
}

public class Aiguillage extends Troncon {}

public class Rail extends Troncon {}

public class Train {
    protected int code;
    private Gare vientDe;
    private Gare vaVers;
    private Troncon CirculeSur;
    private Troncon seDirigeVers;
}
    
```

jean-philippe.babau@univ-brest.fr

Exemple UML (éditeur Topcased)

UBO

```

classDiagram
    package TrainMiniature
    class Gare {
        #nom : String
    }
    class Train {
        #code : Integer
    }
    class Reseau
    class Troncon {
        +estRelieA { unique }
    }
    class Rail
    class Aiguillage

    Gare "1" -- "*" Reseau
    Train "1" -- "*" Gare : +vientDe
    Train "1" -- "*" Gare : +vaVers
    Train "1" -- "*" Troncon : +seDirigeVers
    Train "1" -- "*" Troncon : +circuleSur
    Reseau "1" -- "*" Troncon
    Troncon "1" -- "2..3" Troncon : +estRelieA { unique }
    Troncon <|-- Rail
    Troncon <|-- Aiguillage
    
```

jean-philippe.babau@univ-brest.fr

UBO

Différences modèle/code

jean-philippe.babau@univ-brest.fr 7

UBO

Modèle -> code

- " Générer du code
 - . Java, C++, δ
 - . *Reverse*
 - . Suivi des évolutions
- " Générer du code lisible
 - . Relecture de code
 - . Modification du code

jean-philippe.babau@univ-brest.fr 8

UBO

Exemple : code Java généré

The screenshot shows the Eclipse IDE with the following details:

- Package Explorer:** Shows a project structure with packages like `org.rtsimex.usecase.robot` and `org.rtsimex.usecase.robot.Componer`.
- Editor:** Displays the source code for `Reseau.java`. The code includes:


```

      /** TrainMiniature/Reseau.java
       * File generated from the Reseau uml Class
       * Generated by the Acceleo UML 2.1 to Java generator module (Obeco)
       * $ Date : 03/01/11 18:14:23 (3 janvier 2011) $ */
      package TrainMiniature;

      import java.util.HashSet;
      import TrainMiniature.Gare;
      import TrainMiniature.Troncon;
      import TrainMiniature.Troncon;
      // Start of user code to add imports for Reseau
      // End of user code

      /** Description of the class Reseau */

      public class Reseau {
          private HashSet<Gare> gare;
          public Troncon troncon;
          private HashSet<Troncon> troncon;
          // Start of user code to add fields for Reseau
          // End of user code
          /** Constructor. */
          public Reseau() {
              // Start of user code for constructor Reseau
              super();
              // End of user code
          }
          /** Set a value to attribute gare @param gare. */
      }
      
```

UBO

Exemple : code Java généré

“ Structure en classes, Setter, Getter, déclaration des opérations

The screenshot shows the Eclipse IDE with the following details:

- Package Explorer:** Shows a project structure with packages like `Test2` and `tests`. The `TrainMiniature` package contains files like `Aiguillage.java`, `Gare.java`, `Rail.java`, `Reseau.java`, `Train.java`, and `Troncon.java`.
- Editor:** Displays the source code for `Reseau.java`. The code includes:


```

      /** Set a value to attribute gare @param gare. */
      public void setGare(HashSet<Gare> gare) {
          this.gare = gare;
      }
      /** Add a gare to the gare collection.
       * @param gare_elt Element to add. */
      public void addGare(Gare gare_elt) {
          this.gare.add(gare_elt);
      }
      /** Remove a gare to the gare collection.
       * @param gare_elt Element to remove */
      public void removeGare(Gare gare_elt) {
          this.gare.remove(gare_elt);
      }
      /**
       * Return gare.
       * @return gare
       */
      public HashSet<Gare> getGare() {
      }
      
```

UBO

Le diagramme de classe en 4 points

- " Vue statique
 - . Vue structurelle
 - . Classes, relations, contraintes, commentaires
- " Diagramme le plus connu et le plus utilisé d'UML
- " Liens avec les langages de programmation objet
 - . JAVA, C++
- " Utilisation en développement logiciel
 - . Expression des besoins : modèle du domaine
 - . Conception : architecture logicielle
 - . Construction
 - " Architecture du code
 - " Outils de génération de code

jean-philippe.babau@univ-brest.fr 11

UBO

Plan

- " Introduction aux diagrammes de classe
- " Description des diagrammes de classe
 - . Concepts de base
 - . Concepts avancés
- " « Bien » faire des diagrammes de classe

jean-philippe.babau@univ-brest.fr 12

UBO

Les classes

Définition de l'OMG
 "A class describes a set of objects that share the same specifications of features, constraints, and semantics"

- " Descripteur d'objets qui partagent les mêmes caractéristiques
 - . Structure, comportement, contraintes, sémantique
- " Représentation graphique usuelle
 - . Compartiment du haut : nom et stéréotypes
 - . Compartiment du milieu : liste des attributs
 - . Compartiment du bas : liste des opérations

Train
code : Integer vitesse : Integer
Demarre() Stop()

jean-philippe.babau@univ-brest.fr

13

UBO

Les éléments d'une classe

- " **Classe**
 - . **Nom**
 - . **Classe *abstraite* (pas d'objets) ou concrète**
 - . **Stéréotype**
 - " Lien avec un concept défini dans un profil UML
 - " Réutilisation de concepts existants
 - . **Visibilité : public (+), private (-), protected(#), package(-)**
- " **Vues**
 - . Plus ou moins détaillées : la vue graphique reste une vue
 - . Plusieurs vues d'une même classe dans plusieurs diagrammes

<div style="border: 1px solid black; padding: 2px; margin-bottom: 5px;">Window</div> <div style="border: 1px solid black; padding: 2px;"> size: Area visibility: Boolean display() hide() </div>	<div style="border: 1px solid black; padding: 2px; margin-bottom: 5px;">Window</div> <div style="border: 1px solid black; padding: 2px;"> + size: Area = (100, 100) # visibility: Boolean = true + defaultSize: Rectangle - xWin: XWindow display() hide() - attach(xWin: XWindow) </div>	<div style="border: 1px solid black; padding: 2px; margin-bottom: 5px;">Window</div> <div style="border: 1px solid black; padding: 2px;"> public size: Area = (100, 100) defaultSize: Rectangle protected visibility: Boolean = true private xWin: XWindow public display() hide() private attach(xWin: XWindow) </div>
---	---	--

Extrait de « Unified Modeling Language: Superstructure »
version 2.1.1

jean-philippe.babau@univ-brest.fr

14

UBO

Les éléments d'une classe

- ~ **Attribut**
 - . **Nom**
 - . **Attribut d'objet** (*IsStatic = false*) ou de classe (*IsStatic = true*)
 - . **Multiplicité** (*Lower Upper*)
 - . **Visibilité** : public (+), private (-), protected(#), package(~)
 - . **Type**
 - ~ types primitifs ou autre classe
 - ~ *Boolean, Integer, String, UnlimitedNatural* (entier positif, infini : *)
 - . **Valeur initiale** (*Default*)
 - . **Attribut dérivé** (*IsDerived = true*) : la /valeur est calculée en fonction des autres caractéristiques de la classe

Train
+ <i>Compagnie</i> : <i>String</i>
<i>nom</i> : <i>String</i>
- <i>vitesse</i> : <i>Integer = 0</i>

- ~ *visibilité nom : type [multiplicité] = valeurInitiale {propriétés}*

jean-philippe.babau@univ-brest.fr

15

UBO

Les éléments d'une classe

- ~ **Opération**
 - . **Nom**
 - . **Visibilité** : public (+), private (-), protected(#), package(~)
 - . **Liste de paramètres**
 - ~ **Nom de l'argument**
 - ~ **Direction** : *in* *out* *inout*
 - ~ **Type**
 - ~ **Valeur par défaut**
 - . **Type de retour**
 - . **Propriétés**
 - ~ *Abstraite* (*IsAbstract = true*)
 - ~ *Requête sur l'objet* (*IsQuery = true*)
 - ~ *Opération de classe* (*IsStatic = true*)
 - . **Comportement**
 - ~ *Method* : implémentation de l'opération
 - ~ *Pré-post conditions* avec OCL
 - . **Concurrence** : sequential, guarded, concurrent

Train
+ <i>Demarre</i> (<i>in vitesseMax:Integer</i>)

- ~ *visibilité nom(listeArguments) : typeRetour {propriétés}*

jean-philippe.babau@univ-brest.fr

16

UBO

Les relations

Association

- Les instances de la classe sont liées
- Les instances peuvent communiquer directement

Composition

- « A est composé de B »
- A est créé « avec » B
- Si A disparaît, B disparaît
- Si B disparaît, A peut ne plus être valide
- Impact sur les constructeurs

Généralisation

- Les instances de la classe sont des instances de la super-classe
- Notion d'héritage pour les langages de programmation

jean-philippe.babau@univ-brest.fr

17

UBO

Les éléments d'une relation

Nom de la relation

- Forme verbale

Les membres liés

- 2 ou plus
- Nom de chaque rôle
 - Attribut dans le code généré
- Multiplicité
- Navigabilité
 - bidirectionnelle
 - Monodirectionnelle : Source et destination

Visibilité : public (+), private (-), protected(#), package(~)

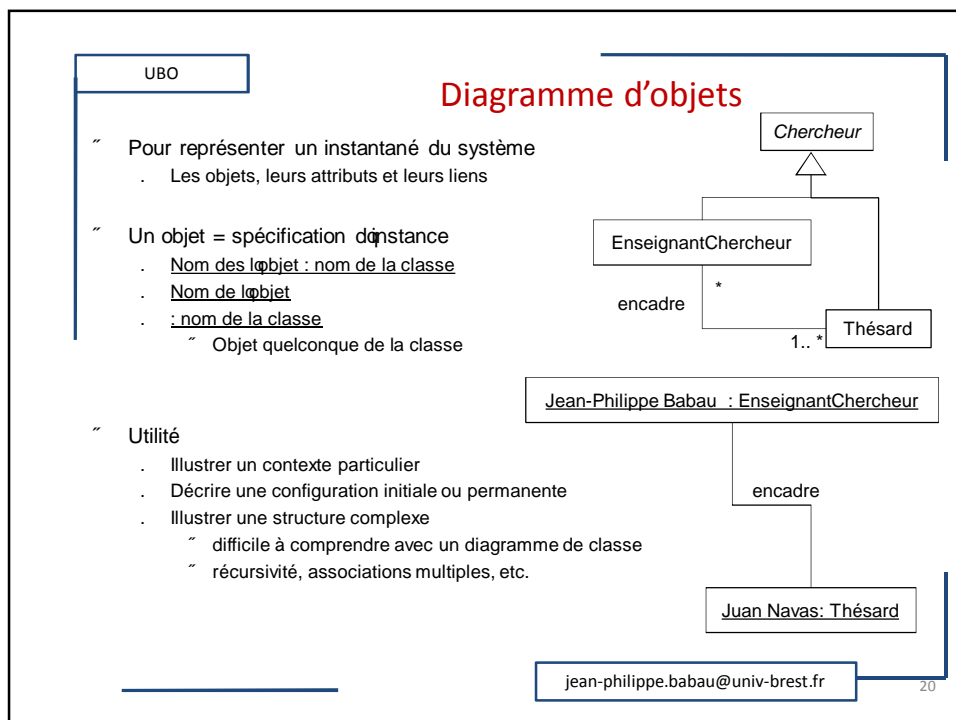
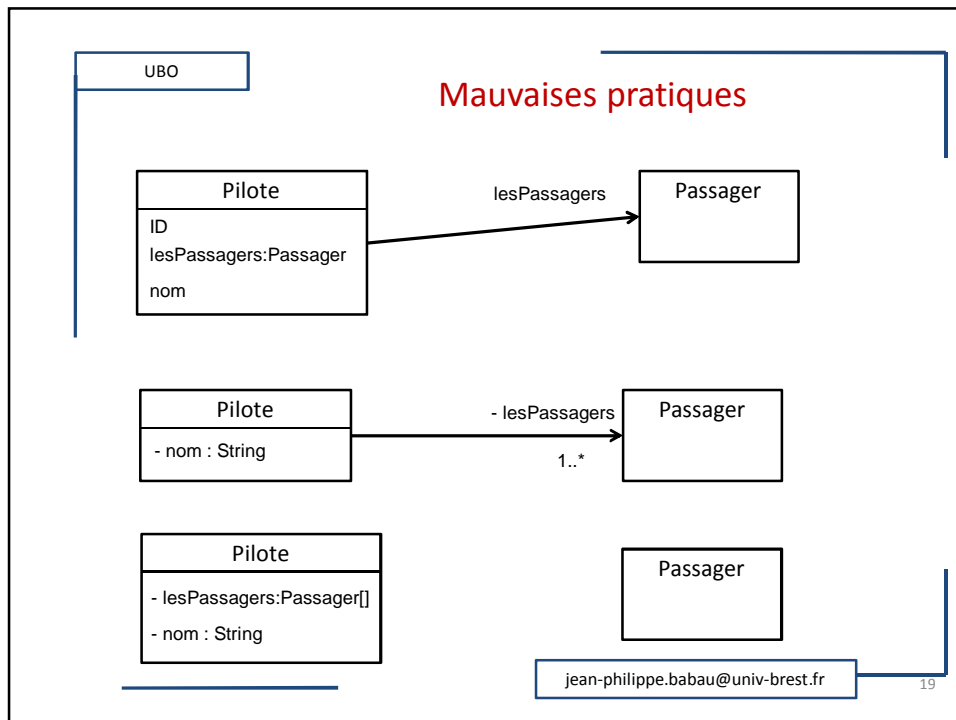
Relation dérivée (*IsDerived = true*) : il existe un autre chemin pour associer les deux classes

Propriétés

- {ordered}
- {unique} set ou bag

jean-philippe.babau@univ-brest.fr

18



UBO

Exemples

“ Vecteur et télévision

jean-philippe.babau@univ-brest.fr

21

UBO

Exemple (issu de l'OMG)

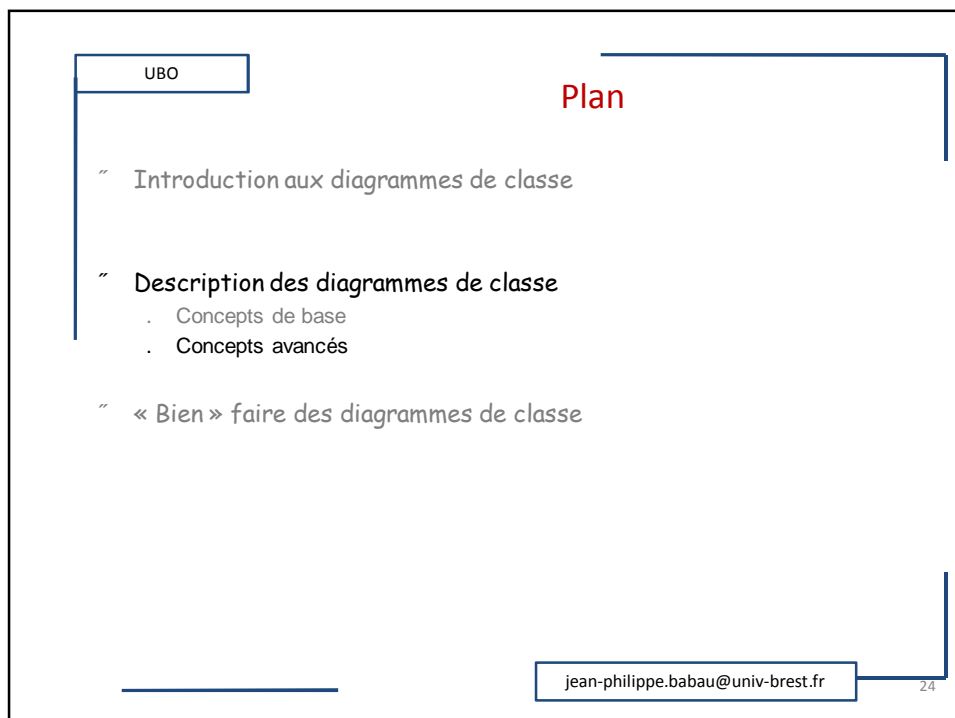
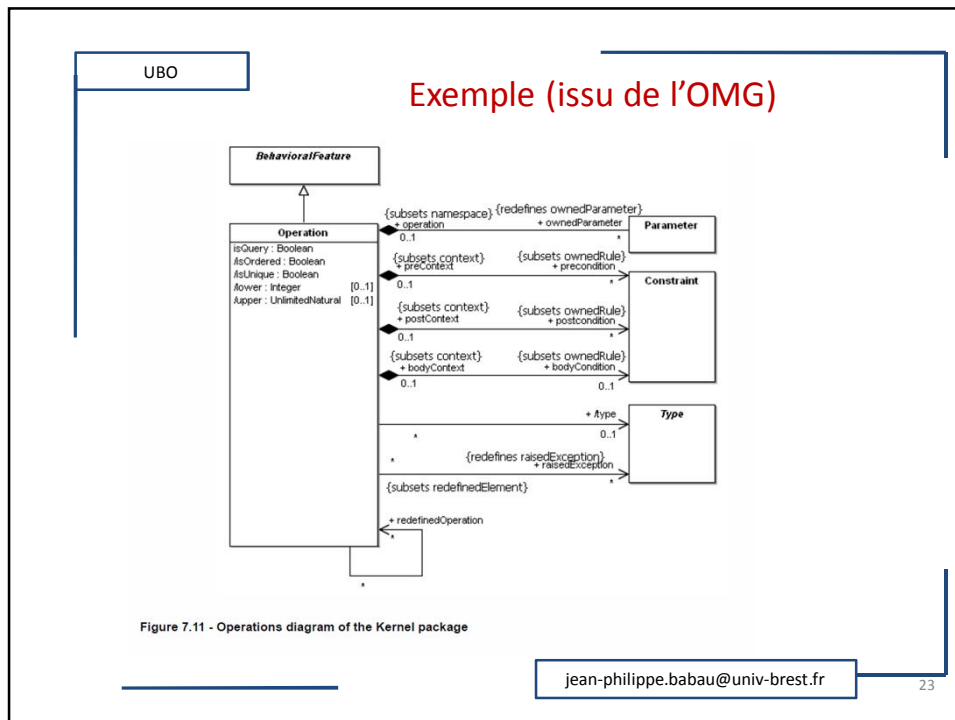
```

classDiagram
    class RedefinableElement
    class Classifier {
        + MultiplicityElement *
        + TypedElement *
        + Namespace *
        + BehavioralFeature *
        + Feature *
        + StructuralFeature *
        + Parameter *
        + Type *
        + ValueSpecification *
    }
    class Feature {
        + isSubclass Boolean
    }
    class BehavioralFeature
    class Namespace
    class Parameter {
        + direction ParameterDirectionKind
        + default String
    }
    class Type
    class ValueSpecification {
        + defaultValue
    }
    class MultiplicityElement
    class TypedElement
    class StructuralFeature {
        + isReadOnly Boolean
    }
    class ParameterDirectionKind {
        in
        out
        return
    }

    RedefinableElement <|-- Feature
    Classifier <|-- BehavioralFeature
    Classifier <|-- MultiplicityElement
    Classifier <|-- TypedElement
    Classifier <|-- StructuralFeature
    Classifier <|-- Parameter
    Classifier <|-- Type
    Classifier <|-- ValueSpecification
    Feature <|-- BehavioralFeature
    Feature <|-- Parameter
    Feature <|-- Type
    Feature <|-- ValueSpecification
    Classifier "1" -- "*" Feature
    Classifier "1" -- "*" Parameter
    Classifier "1" -- "*" Type
    Classifier "1" -- "*" ValueSpecification
    Classifier "1" -- "*" MultiplicityElement
    Classifier "1" -- "*" TypedElement
    Classifier "1" -- "*" StructuralFeature
    Classifier "1" -- "*" Namespace
    Feature "0..1" -- "*" Parameter : + ownerFormalParam (subsets namespace)
    Feature "0..1" -- "*" Type : + raisedException
    Parameter "0..1" -- "*" ValueSpecification : + owningParameter
    Parameter "0..1" -- "*" ValueSpecification : + default value
    
```

jean-philippe.babau@univ-brest.fr

22



UBO

Les stéréotypes

- “ UML définit les concepts de base pour la modélisation
 - . Diagrammes de classes, d'activité, de cas d'utilisation, etc.
- “ Les concepts spécifiques sont définis via les profils
 - . Ensemble de stéréotypes organisés sous forme de diagrammes de classe
 - “ Sémantique et contraintes pour chaque stéréotype
 - . Modélisation de niveau méta-modélisation
- “ Stéréotype applicable à tout élément d'un diagramme UML
 - . Et donc aux éléments d'un diagramme de classe
- “ Notation
 - . <<nom du stéréotype>>

jean-philippe.babau@univ-brest.fr 25

UBO

Exemple issu de l'OMG, profil MARTE

Profil MARTE, package HRM

```

classDiagram
    class HwResource["« stereotype » HwResource"] {
        frequency: NFP_Frequency [0..1]
    }
    class HwMemory["« stereotype » HwMemory"] {
        memorySize: NFP_DataSize
        addressSize: NFP_DataSize
        timings: Timing [*]
    }
    class HwRAM["« stereotype » HwRAM"] {
        organization: MemoryOrganization
        isSynchronous: NFP_Boolean
        isStatic: NFP_Boolean
        isNonVolatile: NFP_Boolean
        repl_Policy: Repl_Policy
        writePolicy: WritePolicy
    }
    HwRAM --|> HwMemory
    HwMemory --|> HwResource
          
```

Application du profil, niveau classe

```

« hwRAM » SDRAM
{
  isSynchronous = true,
  isStatic = false,
  isNonVolatile = false
}
burstLengths: NFP_Natural [1..*]
burstTypes: BurstType [1..*]
refreshRate: RefreshRate
refreshModes: RefreshMode [1..*]
          
```

Application du profil, niveau instance

```

« hwRAM » K4S641632H : SDRAM
{
  frequency = 166 MHz,
  memorySize = 64 MB,
  addressSize = 22 bit,
  organization = (4096, 256, 4, 16 bit),
  timings = ({('CAS', 'CAS latency', 2 CLK), ('RAS', 'row active time', 18 ns)})
}
burstLengths = 1, 2, 4, 8, 4K
burstTypes = sequential, interleave
refreshRate = (4K, 64 ms)
refreshModes = CAS#beforeRAS#
          
```

jean-philippe.babau@univ-brest.fr 26

UBO

Les interfaces

- " Stéréotype <<interface>>
 - . Notion proche de la notion d'interface Java
- " Applicable à une classe
 - . Définit une liste de attributs et d'opérations
- " Réalisé par au moins une classe
 - . Relation spécifique de réalisation

```

classDiagram
    class List {
        <<interface>>
    }
    class ArrayList
    List ..|> ArrayList
          
```

jean-philippe.babau@univ-brest.fr

27

UBO

Interfaces et connexions

- " Exprimer les dépendances entre classes
- " Découpler l'interface des réalisations

ReseauFerroviaire

- lesTrains : Train [*]

+AvanceTrain(in numero : Integer)
+StopTrain(in numero : Integer)

ReseauFerroviaire

- lesTrains : Train [*]

+AvanceTrain(in numero : Integer)
+StopTrain(in numero : Integer)

<<interface>>
Train

+Demarre()
+Stop()

TrainElectrique

+Demarre()
+Stop()
-freine()

TrainElectrique

+Demarre()
+Stop()
-freine()

use

jean-philippe.babau@univ-brest.fr

28

UBO

Plan

- " Introduction aux diagrammes de classe
- " Description des diagrammes de classe
- " « Bien » faire des diagrammes de classe

jean-philippe.babau@univ-brest.fr 29

UBO

Exemples

- " Classe cercle
- " Classe sémaphore

jean-philippe.babau@univ-brest.fr 30

UBO

Les éléments d'une classe

- ~ Nom de classe
 - . Singulier
 - . Commence par une majuscule
- ~ Découpage en classes
 - . Une classe = **UN** concept identifié : principe d'abstraction
 - ~ Issu du domaine pour les concepts clés
 - ~ Encapsulation : une propriété est dans UNE classe
 - . **Attention**, pas de : « A et B représente ou manipule la même information »
 - ~ Faible couplage : des objets ne sont pas interdépendants
 - . **Attention**, pas de : « A a besoin de B qui a besoin de A qui a besoin de B »
- ~ Modélisation orientée donnée
 - . Les attributs modélisent un objet type
 - . Opérations : *setter* et *getter*, calculs sur les données de la classe
- ~ Modélisation orientée services
 - . Regroupement de services
 - . Encapsulation de données pour rendre les services

jean-philippe.babau@univ-brest.fr

31

UBO

Structuration en classe

- ~ Structuration des concepts
 - . Trouver les classes via les concepts (noms) du domaine étudié
 - . Les associations correspondent souvent à des verbes
 - ~ Organise, pilote, contribue à, contient, etc.
 - . Les attributs correspondent souvent à des substantifs, ou des groupes nominaux
 - ~ la couleur d'une voiture, le montant d'une transaction
 - ~ Attention, les attributs sont souvent des relations cachées entre classes
 - . Les adjectifs et les valeurs correspondent souvent à des valeurs d'attributs
 - ~ Bleu, 100 euros

jean-philippe.babau@univ-brest.fr

32

UBO

Structuration en classe

- “ Processus itératif
 - . Classes : les concepts du domaine
 - . Relations : les relations
 - . Attributs et/ou opérations : détailler les concepts
 - . Organiser et simplifier le modèle
 - “ Éliminer les classes redondantes
 - . **Classes qui ont les mêmes attributs, relations et opérations**
 - “ Utiliser l'héritage
 - . **Attributs et relations communes à plusieurs classes**
 - . Itérer et raffiner le modèle
 - “ Un modèle est rarement correct dès sa première construction !!!

jean-philippe.babau@univ-brest.fr

33

UBO

Les relations

- “ Tout objet doit être relié
 - . Un message A peut communiquer directement avec l'objet B
=> une association de A vers B
- “ Nommage
 - . Pour les associations
 - . Pas pour la composition, la généralisation et l'agrégation
 - . Pour les dépendances : utilisation de noms prédéfinis
- “ Multiplicité
 - . Nombre possible de relations pour une instance
 - . Dans les deux sens
- “ Directions
 - . Pourquoi limiter la navigabilité à un sens ?

jean-philippe.babau@univ-brest.fr

34

UBO

Les relations

- “ Composition
 - . Association très forte
 - . Objets interdépendants
- “ Généralisation (héritage) multiple
 - . Autorisé en UML
 - . À limiter
 - “ **Attention** : pas de caractéristiques héritées possédant le même nom

jean-philippe.babau@univ-brest.fr

35

UBO

La généralisation

- “ Généralisation
 - . Modélisation métier et conception
 - “ Le concept lié à la super-classe est plus général
 - . Codage
 - “ Héritage des attributs et des opérations
 - “ Structuration du code
- “ Principes à respecter « B hérite de A »
 - . Principe de substitution
 - “ Toutes les propriétés (attributs, relations, contraintes) de A sont correctes pour la classe B
 - . B est une sorte de A
 - “ Toutes les instances de B sont des instances de A

jean-philippe.babau@univ-brest.fr

36

UBO

Quand définir une super-classe

- “ Concept général et générique
- “ Plusieurs classes ont la même propriété
 - . Attribut *identifiant* dans plusieurs classes
 - . Création de la classe *ElementIdentifie* avec l'attribut *identifiant*
 - . Les classes héritent de la classe *ElementIdentifie*
- “ Détecter de « faux » attributs de type énumération
 - . Une classe possède un attribut *Value* qui a deux valeurs possibles
 - . Créer une classe abstraite sans l'attribut *Value*
 - . Créer deux classes qui hérite de la classe abstraite, chacune modélisant un des deux concepts implicitement modélisé par l'attribut *Value*
- “ Peu de super-classes sans propriétés

jean-philippe.babau@univ-brest.fr

37

UBO

Voiture et autoradio

Voiture

Conducteur

Autoradio
On()
Off()
Select(int number)
VolumeUp()

jean-philippe.babau@univ-brest.fr

Exemple : contrôle d'un robot

Structure

- mise en place des 3 couches entre le contrôle et l'environnement (ici le robot)

Mauvais positionnement de l'utilisateur

- Recommandation : insérer 3 couches entre l'utilisateur et le contrôle

jean-philippe.babau@univ-brest.fr 39

Exemple

jean-philippe.babau@univ-brest.fr 40

UBO


Critiques

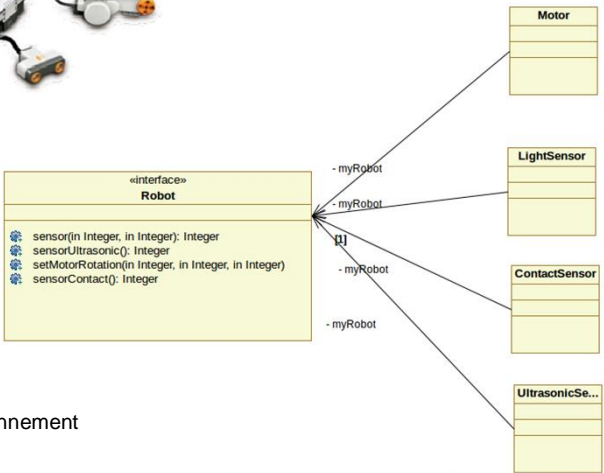
- " Intérêt du modèle
- " Points faibles du modèle

jean-philippe.babau@univ-brest.fr
41

UBO

Exemple : le robot





```

classDiagram
    class Robot {
        <<interface>>
        sensor(in Integer, in Integer) Integer
        sensorUltrasonic() Integer
        setMotorRotation(in Integer, in Integer, in Integer)
        sensorContact() Integer
    }
    class Motor
    class LightSensor
    class ContactSensor
    class UltrasonicSe...
    Robot <|-- Motor
    Robot <|-- LightSensor
    Robot <|-- ContactSensor
    Robot <|-- UltrasonicSe...
    
```

- " Vue de l'environnement

jean-philippe.babau@univ-brest.fr
42

UBO

Exemple : contrôle et données de haut-niveau

```

classDiagram
    class Control
    class Move
    class Rotate
    class Switch
    class Position
    class GoalPosition

    Control "1" -- "1" Move : - myMove
    Control "1" -- "1" Rotate : - myRotate
    Control "1" -- "1" Switch : - mySwitch
    Control "1" -- "1" Position : myPosition
    Control "1" -- "1" GoalPosition : - myGoalPosition
    
```

“ Vue indépendante de l'environnement
 “ Partie générique « portable »

jean-philippe.babau@univ-brest.fr

43

UBO

Exemple : les connexions

```

classDiagram
    class Position
    class LightToPosition
    class LightSensor
    class Switch
    class SensorToSwitch
    class UltrasonicSensor
    class Move
    class MoveToMotor
    class Motor
    class Rotate
    class RotateToMotor

    Position "1" -- "1" LightToPosition : - myPosition
    LightToPosition "1" -- "1" LightSensor : - myL2P
    Switch "1" -- "1" SensorToSwitch : - mySwitch
    SensorToSwitch "1" -- "1" UltrasonicSensor : - myC2S
    Move "1" -- "1" MoveToMotor : - myM2M
    MoveToMotor "1" -- "1" Motor : - myMotor
    Rotate "1" -- "1" RotateToMotor : - myR2M
    RotateToMotor "1" -- "1" Motor : - myMotor
    
```

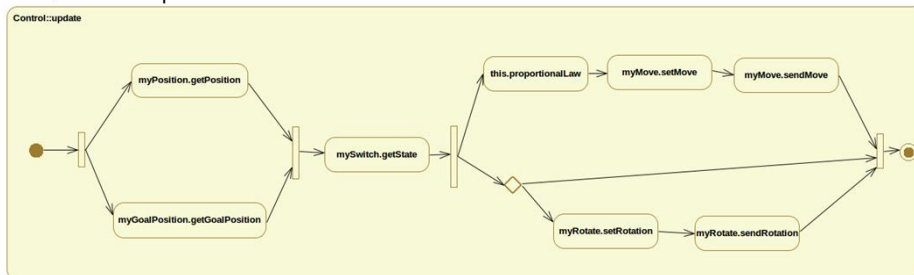
jean-philippe.babau@univ-brest.fr

44

UBO

Exemple : lien entre diagramme de classe et diagramme de comportement

- Comportement sous forme de diagramme d'activité
 - Mise en place d'un PID après acquisition des données et surveillance de l'environnement



- Découplage entre la mise en place de la commande et son envoi
 - Recommandation : regrouper les deux activités
- Pas de découplage entre le contrôle et la surveillance
 - Recommandation : créer deux comportements distincts : et donc 2 opérations

jean-philippe.babau@univ-brest.fr

45

UBO

Modélisation Vs Codage

- Un diagramme de classe UML n'est pas un diagramme de classe Java
- UML : vue graphique de la structure des classes
- Code Java : pas de vue globale
 - Navigation dans les fichiers
- UML : structuration à l'aide des packages
- Code Java : structuration à l'aide des packages et des fichiers
- UML : plusieurs vues possibles
 - Une vue est définie selon un point de vue
- Code Java : une seule vue via les fichiers
- UML : indépendant d'un langage de programmation
- Code Java : lié à Java !!

jean-philippe.babau@univ-brest.fr

46

UBO

Modélisation Vs Codage

- " UML : pas de détail d'implémentation
 - . Une association est une abstraction d'une relation
- " Code : tous les détails d'implémentation
 - . Une association est un tableau, une liste, ...
- " UML : non exécutable
- " Code Java : exécutable
- " UML : le comportement est dans un diagramme différent
 - . Parfois lourd pour un langage algorithmique
- " Code Java : le comportement est décrit de manière textuelle
- " Le diagramme de classe UML est adapté à une vue de haut-niveau de la structuration en classes
 - . Modélisation orientée utilisateur et non orienté code
 - . Modélisation adaptée à la conception
 - . **UML n'est pas un langage de programmation**

jean-philippe.babau@univ-brest.fr

47

UBO

Conclusion

- " Diagramme de classe
 - . Classes (attributs et opérations)
 - . Relations (association, composition, généralisation)
- " Structuration des concepts
 - . Organisation et représentation
 - . Abstraction, encapsulation, faible couplage
- " Mise en œuvre
 - . Processus itératif
- " Codage
 - . Génération de code automatique
 - . Optimisation et précisions manuelles dans le code
 - . Utilisation des bibliothèques

jean-philippe.babau@univ-brest.fr

UBO

Bibliographie

- " OMG et UML
 - . <http://www.omg.org/>
 - . <http://www.uml.org/>
- " Cours de Yannick Prié
 - . <http://liris.cnrs.fr/yannick.prie/ens/09-10/SIMA/index.html>

jean-philippe.babau@univ-brest.fr

49