

UNIVERSITE DE CORSE
2024-2025

Licence ST 3ème année
Option INFORMATIQUE

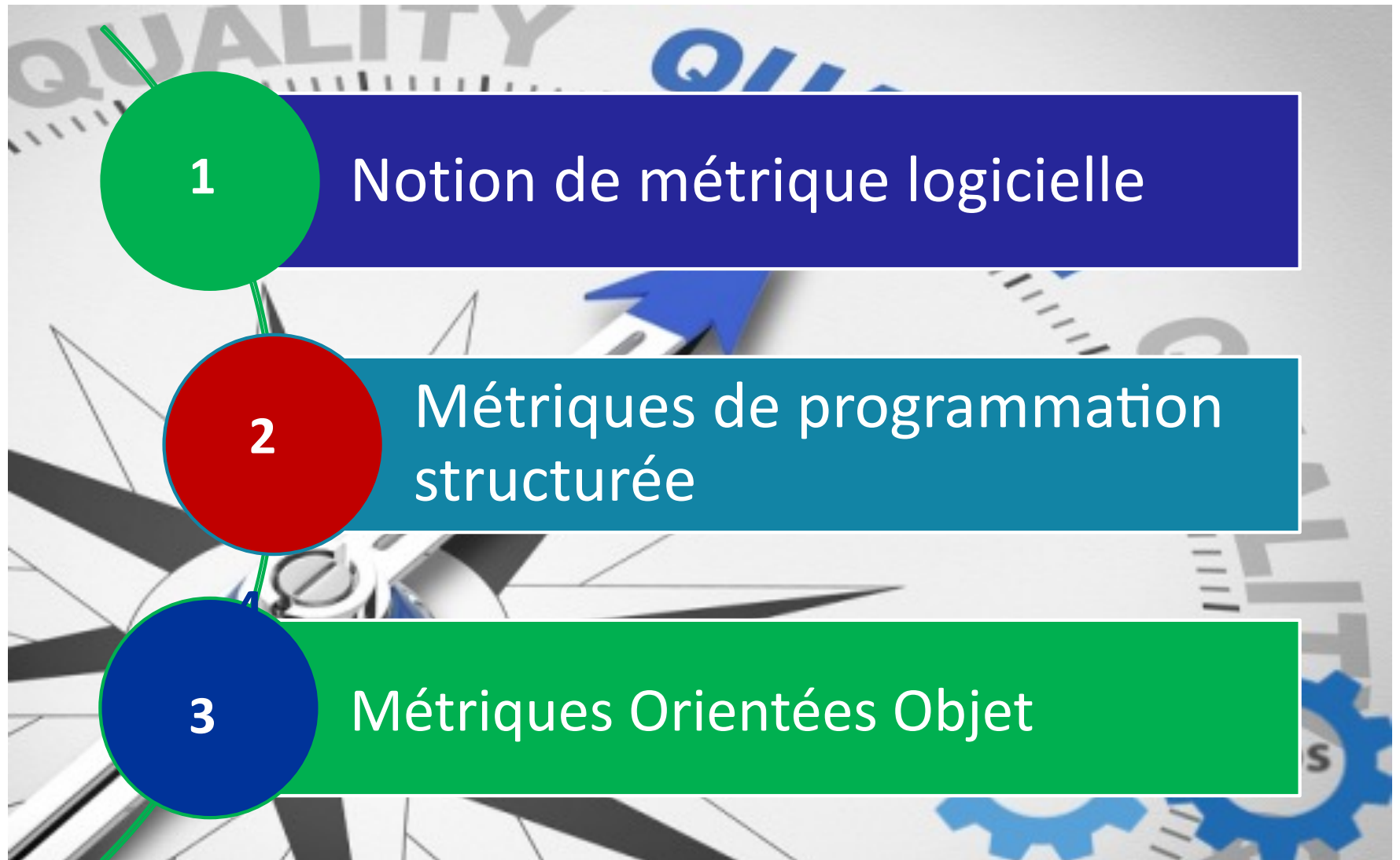
UE Qualité Logicielle et Tests

CH2.3 – Métriques orientées objet

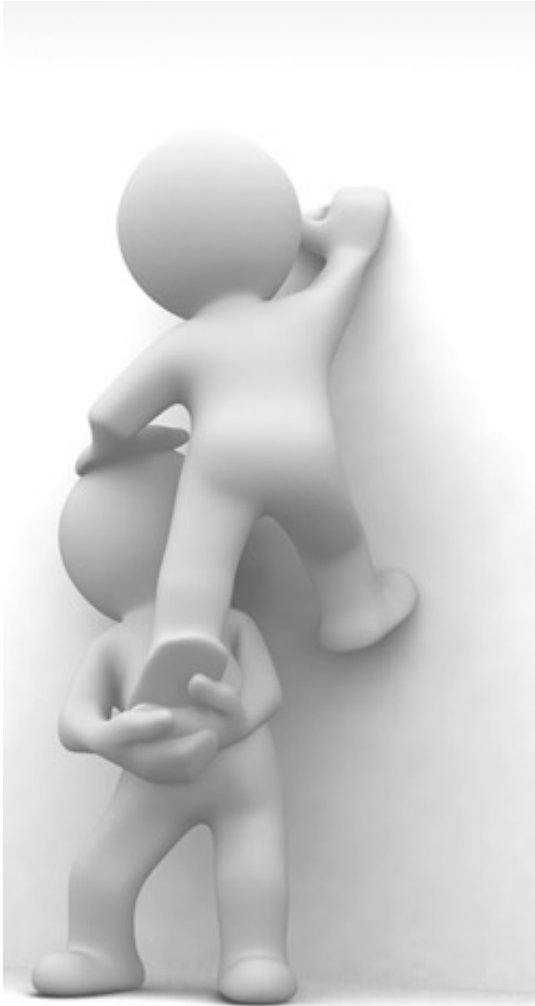


Evelyne VITTORI
vittori@univ-corse.fr

CH2- Métriques logicielles



Objectifs de cette section



- Découvrir les métriques OO et savoir les interpréter:
 - Métriques élémentaires
 - Métriques de couplage et de cohésion
- Comprendre les notions de couplage et de cohésion et leur impact sur la qualité du code

Métriques Orientées Objet

■ Chidamber & Kemerer (1991): 6 métriques de base

- **Couplage-Cohésion**: Coupling between objects (CBO), Lack of Cohesion in Methods (LCOM)
- **Héritage**: Depth of Inheritance Tree (DIT), Number of Children (NOC),
- **Méthodes-complexité**: Response for a Class(RFC), Weighted Methods per Class (WMC)

■ Li & Henry (1993-1995) (Data Abstraction Coupling DAC)

■ Plusieurs catégories de métriques selon leur niveau d'application:

Méthode - Classe - Package - Système



Métriques OO élémentaires



Métriques OO élémentaires

- **NBC** (*NumBer of Classes*) = nombre de classes dans un projet, un package
- **NOA** (*Number Of Attributes*) = nombre d'attributs
- **NSA** (*Number of Static Attributes*) = nombre de variables statiques
- **NOM** (*Number Of Methods*) = nombre de méthodes
- **NSM** (*Number Of Static Methods*) = nombre de méthodes statiques

Métriques liées à l'héritage et l'abstraction

- **NOI** (*Number Of Interfaces*) = nombre d'interfaces
- **RMA** (*Rate of Abstraction*) = pourcentage de classes abstraites et d'interfaces par package
- **DIT** (*Depth of Inheritance Tree*) = profondeur de l'arborescence de classes (niveaux depuis la classe *Object*)

Ces métriques sont très dépendantes du type de logiciel

Métriques liées à l'héritage et l'abstraction

- **NORM** (*Number of Overridden Methods*)= Nombre de méthodes redéfinies
- **SIX** (*Specialization Index*)= indice de spécialisation d'une classe.
 - $SIX = NORM * \text{DIT} / \text{NOM}$

profondeur héritage

nombre de méthodes
 - Pour un projet SIX = moyenne des SIX des classes

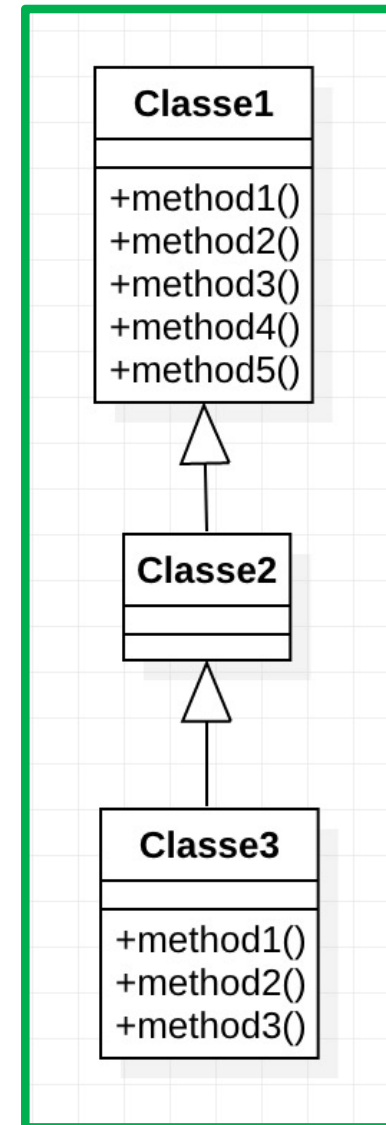
SIX >1,5 la classe redéfinit trop de méthodes ou la profondeur est trop importante

Exercice: métrique SIX

■ Question 1:

Calculez les métriques NORM, DIT, NOM puis la métrique **SIX** pour chacune des classes du diagramme suivant.

Puis pour la métrique SIX du projet.



Exercice: métrique SIX

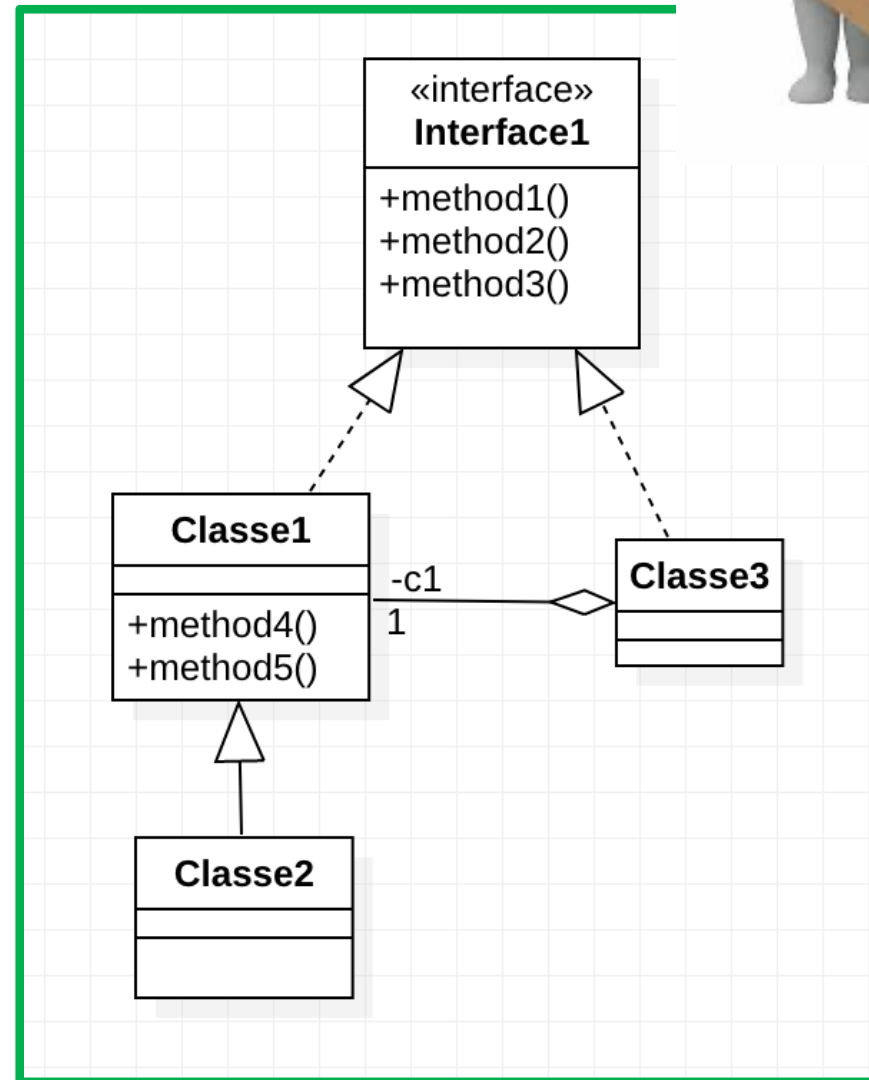


■ Question 2:

Le diagramme a été modifié.

ReCalculez la métrique **SIX** pour chacune des classes du diagramme suivant puis pour le projet.

■ Question 3 : Conclusions?



Métrique de complexité

- **WMC** (*Weighted Methods per Class*) = somme de la complexité cyclomatique de McCabe de toutes les méthodes de la classe
 - $WMC = \sum C_{Ci}$ avec C_{Ci} complexité de la méthode i

Plus WMC est élevé, plus la classe sera complexe à tester et à maintenir

Propositions de valeurs limites

Métriques de niveau « classe »

Metric	Threshold limit	Recommendations above the Threshold limit	Recommendations under Threshold limit
Class Lines of Code (cLOC)	Greater than 500	Class segmentation to more than one Class	No Recommendations
Average number of McCabe's cyclomatic complexity (CC)	Greater than 10	The class is complex	A well-structured Class
Number of Methods (NM).	Greater than 20	Functional examination of the class	No Recommendations
Number of direct Children (NOC)	Greater than 6	High reusable thus requires examination of class carefully because it depends upon a large number of Classes	Indicating no reuse in the class
Number of Methods overridden (NMO).	Greater than 3	The class is complex and difficult to understand	No Recommendations
Weighted Methods per Class(WMC)	Greater than 15	The class is complex	A well-structured Class
Depth of Inheritance Tree (DIT)	Greater than 5	The complexity of the class as a whole is increasing and there is difficulty in calculating the behavior of the class	No Recommendations

Source : « Three Levels Quality Analysis Tool for Object Oriented Programming » , Mustafa Ghanem Saeed1, Kamaran HamaAli Faraj4, Maher Talal Alasaady, Fahad Layth Malallah (IJACSA) International Journal of Advanced Computer Science and Applications, Vol. 9, No. 11, 2018 <https://www.semanticscholar.org/paper/Three-Levels-Quality-Analysis-Tool-for-Object-Saeed-Alasaady/b4993c9c1e3e5627bea29b932c81c096971c1f09>

Propositions de valeurs limites

Métriques de niveau « package »

Metric	Threshold limit	Recommendations above the Threshold limit	Recommendations under Threshold limit
Average Weighted Methods per Class(aWMC)	Greater than 3.	The package is complex	A well-structured package
Average Number of Methods overridden (aNMO).	Greater than 15.	The classes are complex and difficult to understand	No Recommendations
Abstractness – RMA	Greater than (0.5)	Abstract package	Cohesive package
Normalized Distance from Main Sequence- Dn	Greater than (0.5)	The package is unstable	The package is stable

Source : « Three Levels Quality Analysis Tool for Object Oriented Programming » , Mustafa Ghanem Saeed1, Kamaran HamaAli Faraj4, Maher Talal Alasaady, Fahad Layth Malallah (IJACSA) International Journal of Advanced Computer Science and Applications, Vol. 9, No. 11, 2018 <https://www.semanticscholar.org/paper/Three-Levels-Quality-Analysis-Tool-for-Object-Saeed-Alasaady/b4993c9c1e3e5627bea29b932c81c096971c1f09>



Couplage

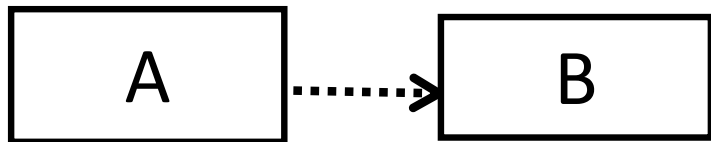
- Qu'est-ce que le couplage?
- Principe de couplage faible
- Métriques d'évaluation du couplage



Qu'est-ce que le couplage?

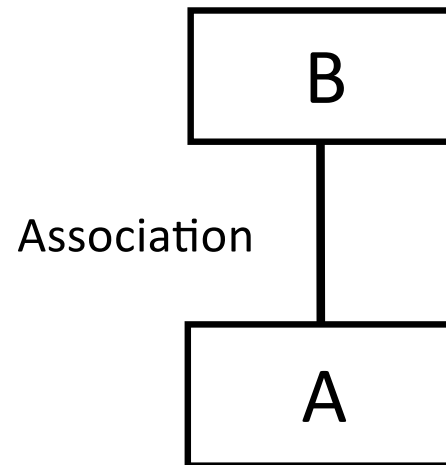
Couplage

Mesure du degré auquel un élément est lié (*prend appui sur/ connaît*) à un autre.

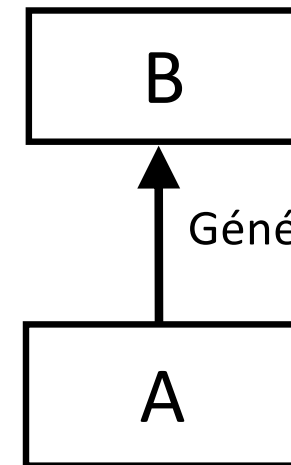


Dépendance (A dépend de B)

- variable locale
- paramètre de méthode
- résultat de méthode
- Invocation de méthode de classe



Association

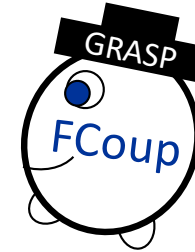


Généralisation



Différents niveaux de couplage

Principe de FAIBLE COUPLAGE



Objectif

Minimiser l'impact des modifications d'une classe sur une (des) autre(s)

Solution

Limiter les dépendances entre classes. Appliquer ce principe pour choisir entre différentes solutions possibles.

Un Couplage faible doit être assuré au niveau des classes à forte probabilité de changement.

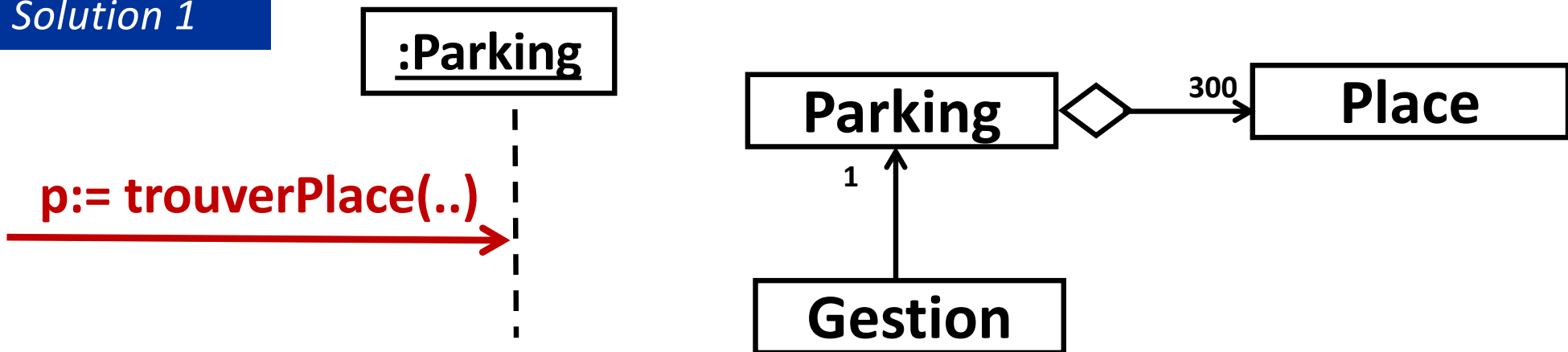
Un couplage fort à des éléments stables n'est pas un problème !!



Principe de FAIBLE COUPLAGE

Exemple Placement de l'opération «trouver place libre» dans un parking. Deux solutions possibles: Choix?

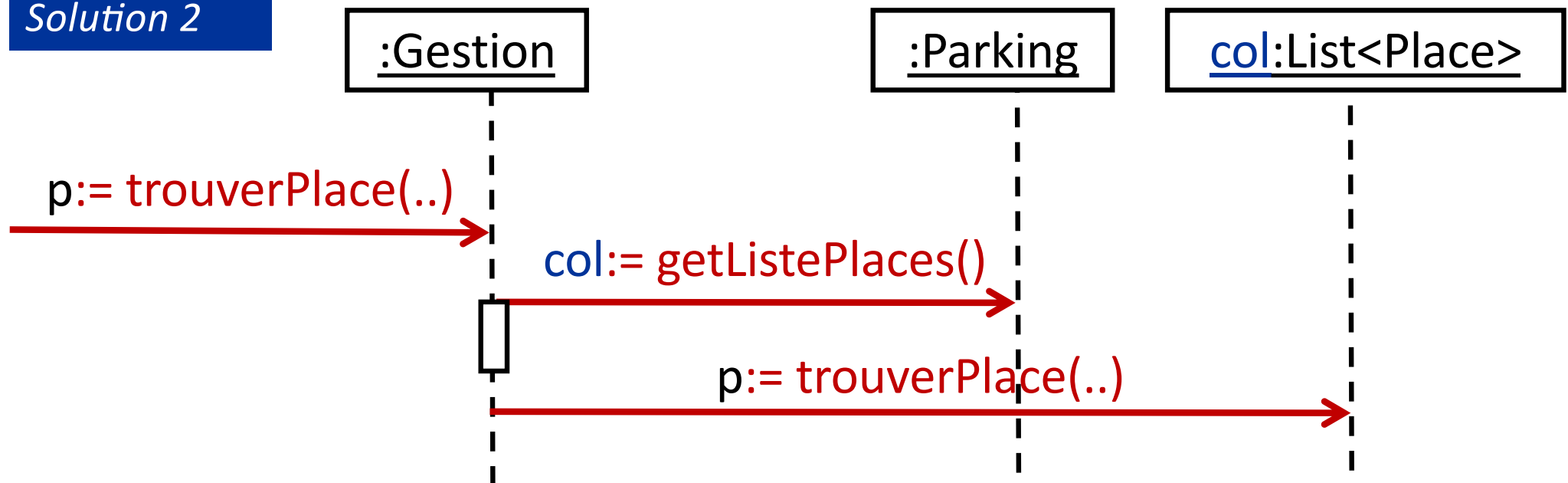
Solution 1



Principe de FAIBLE COUPLAGE

Exemple Placement de l'opération «trouver place» dans un parking.
Choix entre 2 solutions?

Solution 2



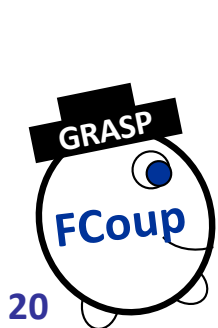
Principe de FAIBLE COUPLAGE

Exemple Choix entre 2 solutions: comparaison du niveau de couplage.

Solution 2 Gestion et Parking sont couplées avec Place



Solution 1 Seule Parking est couplée avec Place



Sans hésitation, j'opte pour la solution 1 car elle n'introduit pas de couplage supplémentaire!

Métriques de couplage

Comment mesurer le niveau de couplage d'un package?

CE=Efferent Coupling

CA=Afferent Coupling

Comment mesurer le niveau de couplage d'une classe?

CBO Coupling between objects

Métriques de couplage

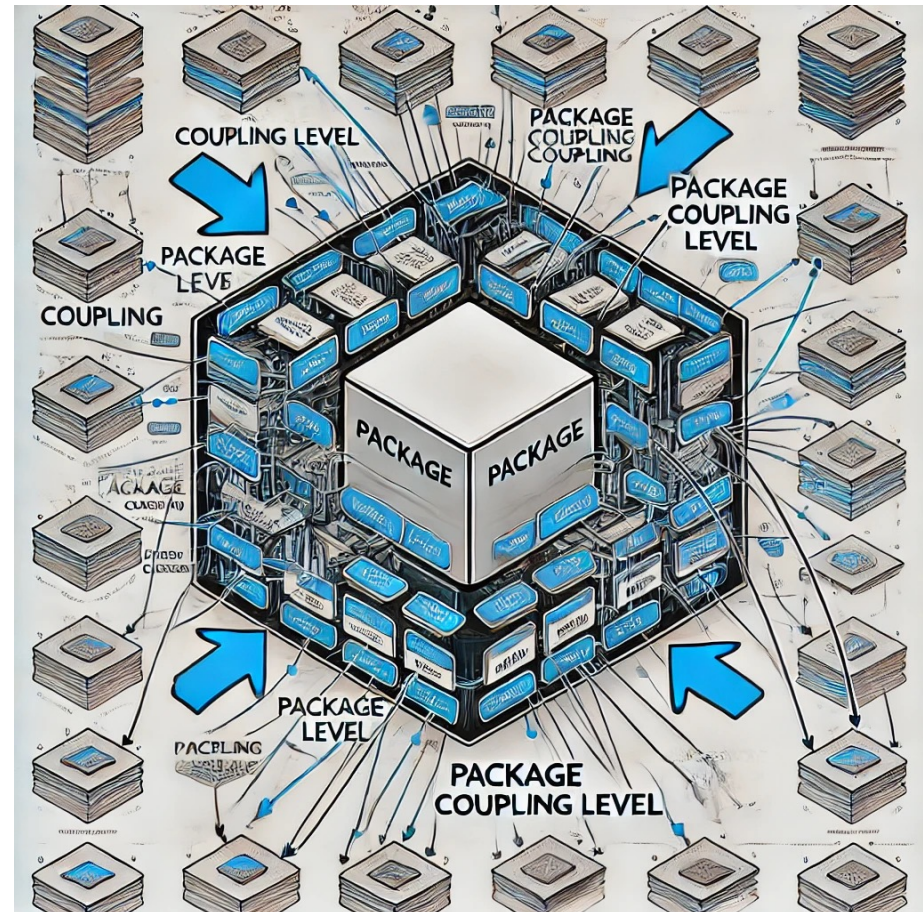
- Mesures du degré de couplage d'une classe C
 - *CBO Coupling Between Objects*
 - nombre de classes utilisées par une classe C (classes associés et classes dont elle dépend)
 - *nCBO (Normalized CBO) $CBO/(N-1)$*
 - N=nombre de classes du package ou projet

nCBO : compris entre 0 et 1
1 = couplage maximum
0 = aucun couplage

L'objectif est de minimiser nCBO

Mesures de couplage d'un package

- Somme des CBO (des nCBO)
- Valeurs Maximum CBO et nCBO
- Moyennes des CBO (et des nCBO)



Exercice: métriques CBO-nCBO

- Calculez les métriques CBO et nCBO des deux diagrammes ci-dessous afin de démontrer que le diagramme 1 possède un niveau de couplage supérieur à celui du diagramme 2.

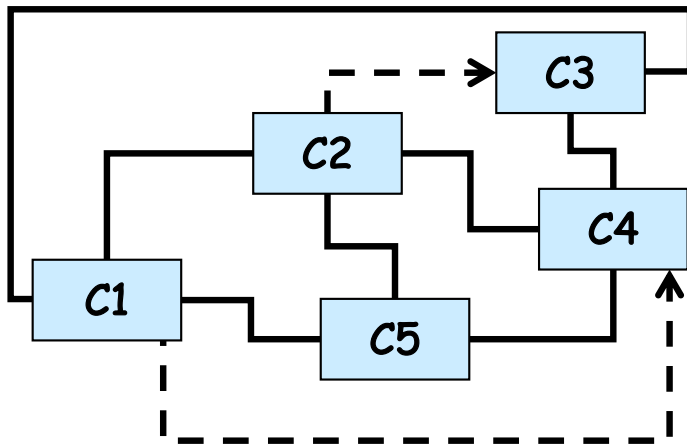


Diagramme 1

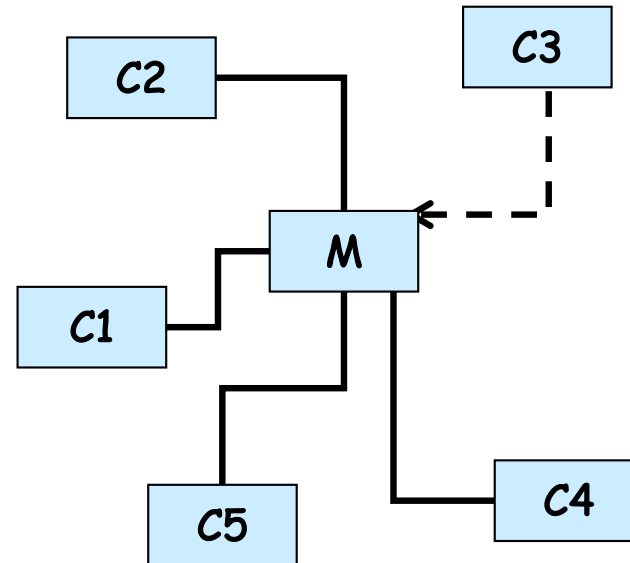


Diagramme 2

Métriques de couplage

Mesures du degré de couplage d'un package

- **CE** (*Efferent Coupling*) =
nombre de classes dans un package qui dépendent d'une classe d'un autre package
- **CA** (*Afferent Coupling*) =
nombre de classes hors d'un package qui dépendent d'une classe dans le package

Si $CE > 50$, le couplage est trop élevé.

Couplage et Instabilité

Indice d'instabilité d'un package

- $RMI = CE / (CA + CE)$

Indicateur de l'instabilité du package c'est-à-dire des dépendances entre packages

RMI: compris entre 0 et 1

1 = package très instable

Toute modification dans un des packages dont il dépend est une menace

0 = stabilité maximum

Conseils

- Essayer de limiter **les packages instables**
- Un effort particulier devra être fait au niveau des tests des packages instables.
- Pour compléter la mesure DMI, il est intéressant d'étudier le **degré d'abstraction**

On calcule :

- RMA = pourcentage de classes abstraites et d'interfaces par package
- Calcul d'une autre Métrique : DMS

Métrique d'équilibre abstraction/instabilité

Distance from the Main Sequence (Normalized Distance)

Une bonne DMS doit être proche de 0

- **DMS = |RMA + RMI - 1|**
 - Mesure de l'équilibre entre le niveau d'abstraction et l'indice d'instabilité
 - Si le package est instable :
 - son niveau d'abstraction compense son instabilité DFS sera bonne

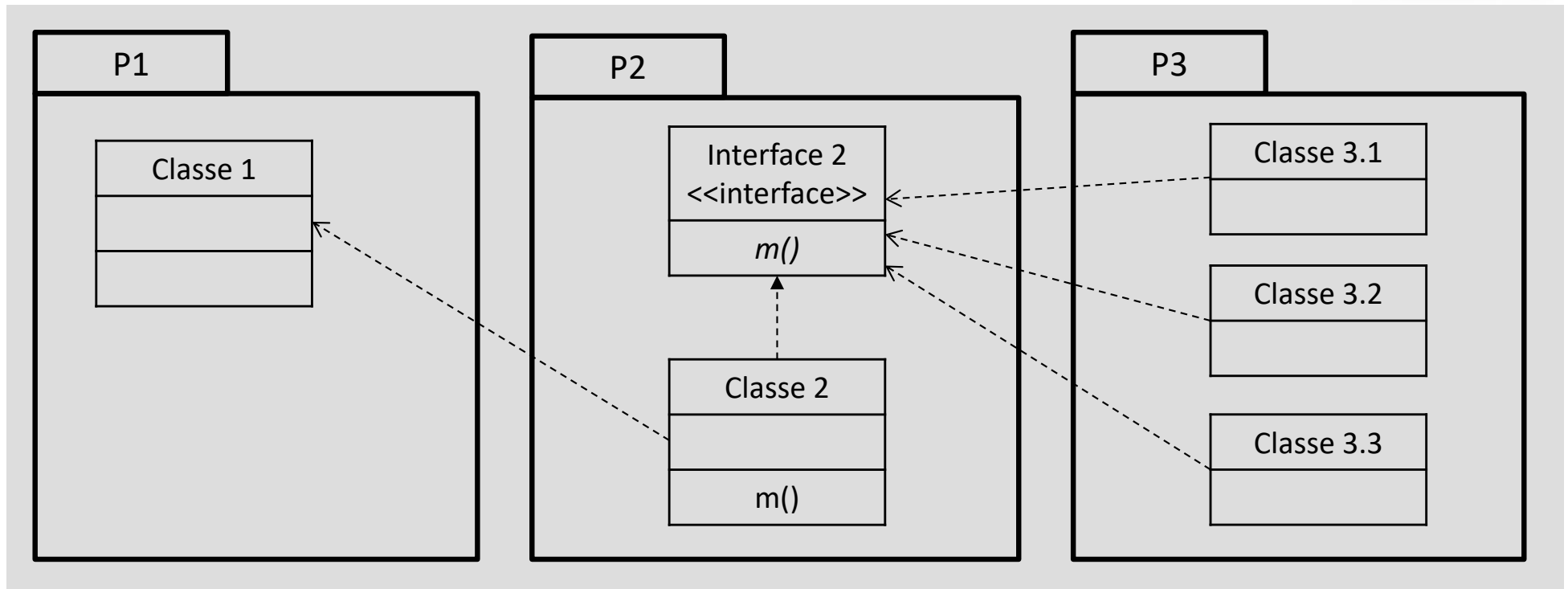
Rappels

- *RMA (Rate of Abstraction)* = pourcentage de classes abstraites et d'interfaces par package
- *RMI (Rate of Instability)*

Métrique d'équilibre abstraction/instabilité

RMI Instabilité	RMA Abstraction	DMS	Interprétation
mauvais	bas	bonne	Le package est instable mais il possède peu d'interfaces : il n'est pas très sollicité donc son instabilité n'est pas problématique
mauvais	élevé	bonne	Beaucoup d'autres packages dépendent de ce package mais il a beaucoup d'interfaces donc le risque est acceptable
bon	bas	mauvaise	La situation est dangereuse car le package ne possède pas beaucoup de classes abstraites et interfaces

Exercice: métriques RMI-DMS



Etudions l'organisation de ces packages.

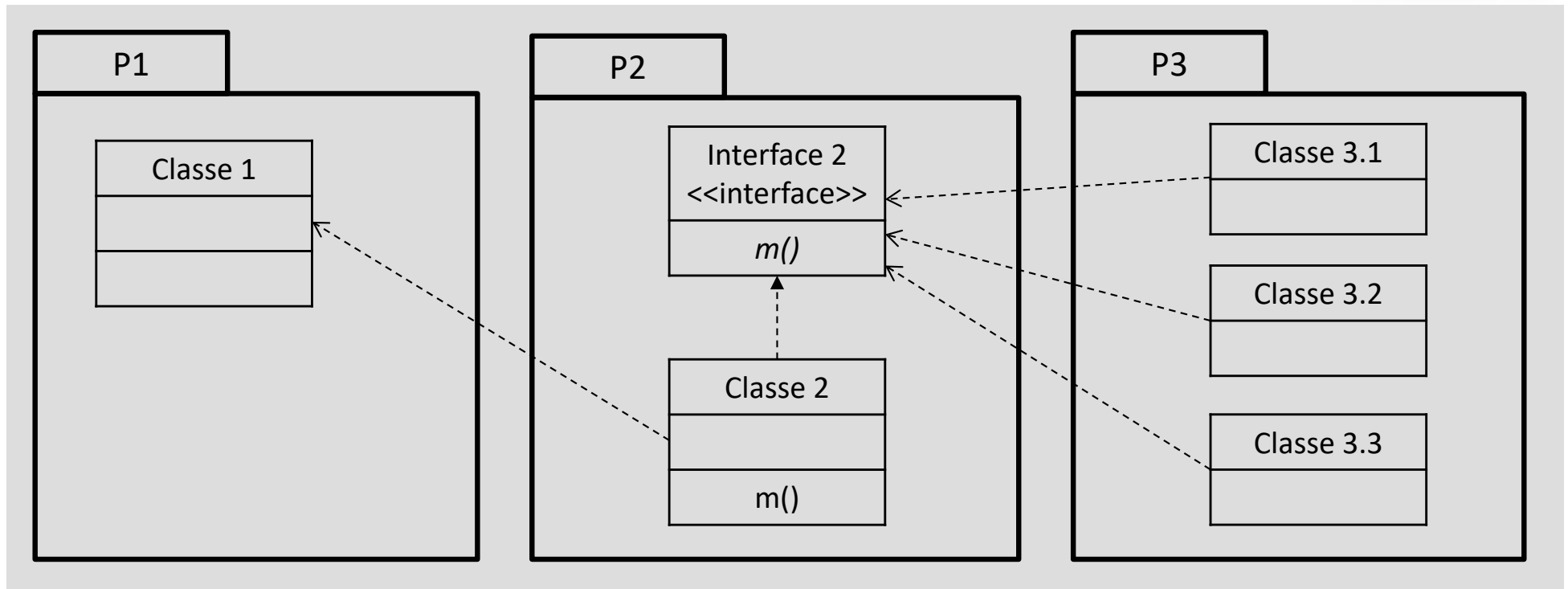
Question 1 - Calculez les métriques **RMI** et **DMS** pour chacun des packages du diagramme ainsi que globalement pour la moyenne des RMI du projet.

Correction Exercice: métrique RMI-DMS



	CE	CA	RMA	RMI	DMS	Diagnostic Stabilité (RMI)	Diagnostic DMS
	nb classes dépendant d'autres packages	nb classes d'ailleurs dépendant du package	Nbclasses abstraites+nb interfaces/nb classes	$CE/(CE+CA)$	$ RMA+RMI-1 $	Stable/Assez stable/Instable	Bonne/correcte/mauvaise
P1							
P2							
P3							
Projet							

Exercice: métriques RMI-DMS



Question 2 – Diagnostic de l'organisation

Quels sont les packages qui posent problème?

Pourriez-vous suggérer des solutions?

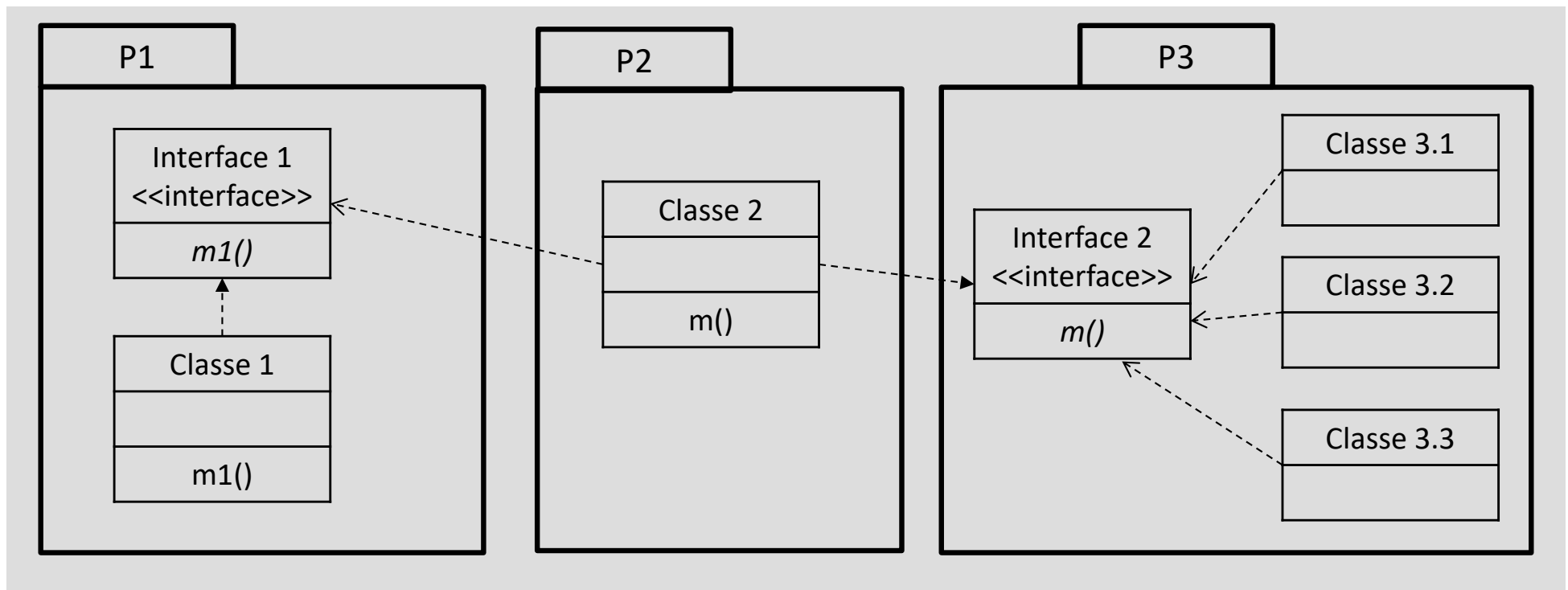
Exercice: métriques RMI-DMS



Question 3 – Réorganisation proposée

Calculez à nouveau les RMI et DMS des packages et du projet après sa réorganisation.

Qu'en pensez-vous? Pour chaque package indiquez s'il a été amélioré.



Correction Exercice: métrique RMI-DMS



	CE	CA	RMA	RMI	DMS	Diagnostic
	nb classes dépendant d'autres packages	nb classes d'ailleurs dépendant du package	Nbclasses abstraites+nb interfaces/nb classes	$CE/(CE+CA)$	$ RMA+RMI-1 $	Stable/Assez stable/Instable DMS Bonne/correcte/mauvaise
P1						
P2						
P3						
Projet						

Correction Exercice: métrique RMI-DMS



	CE	CA	RMA	RMI	DMS	Diagnostic
	nb classes dépendant d'autres packages	nb classes d'ailleurs dépendant du package	Nbclasses abstraites+nb interfaces/nb classes	$CE/(CE+CA)$	$ RMA+RMI-1 $	
P1	0	1	0,5	0	0,5	STABLE et DMS correcte
P2	1	0	0	1	0	INSTABLE mais DMS BONNE
P3	0	1	$\frac{1}{4}=0,25$	0	0	STABLE et DMS BONNE
Projet				Moyenne RMI= $1/3=0,33$ Amélioration		



Cohésion

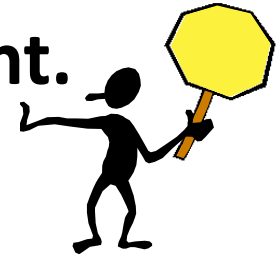
- Qu'est ce que la cohésion?
- Principe de forte cohésion
- Métriques d'évaluation de la cohésion



Qu'est ce que la COHESION?

Cohésion

Mesure de l'étroitesse des liens, de la spécialisation et du volume des responsabilités d'un élément.



Classe fortement cohésive = toutes les méthodes contribuent à un comportement bien délimité

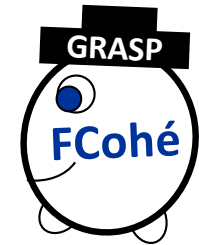
Classe faiblement cohésive = classe exécutant beaucoup de tâches sans liens entre elles ou trop de tâches

Faible
cohésion

Problèmes

Compréhension
Réutilisation
Maintenabilité
Fragilité

Principe de FORTE COHESION

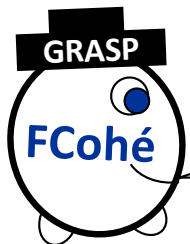


Objectif

S'assurer que les objets restent compréhensibles et faciles à gérer.

Solution

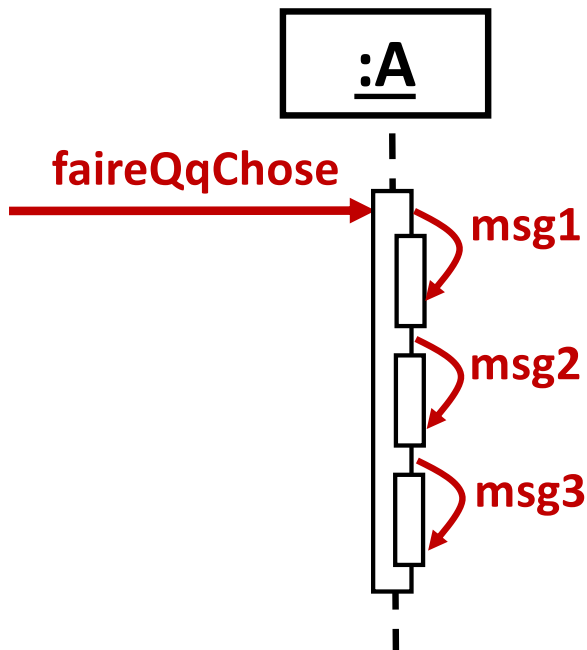
Maintenir la cohésion des différentes classes.
Appliquer ce principe pour choisir entre différentes solutions possibles.



Une classe doit savoir « **déléguer** » des responsabilités pour être efficace!!

Principe de FORTE COHESION

Faible cohésion

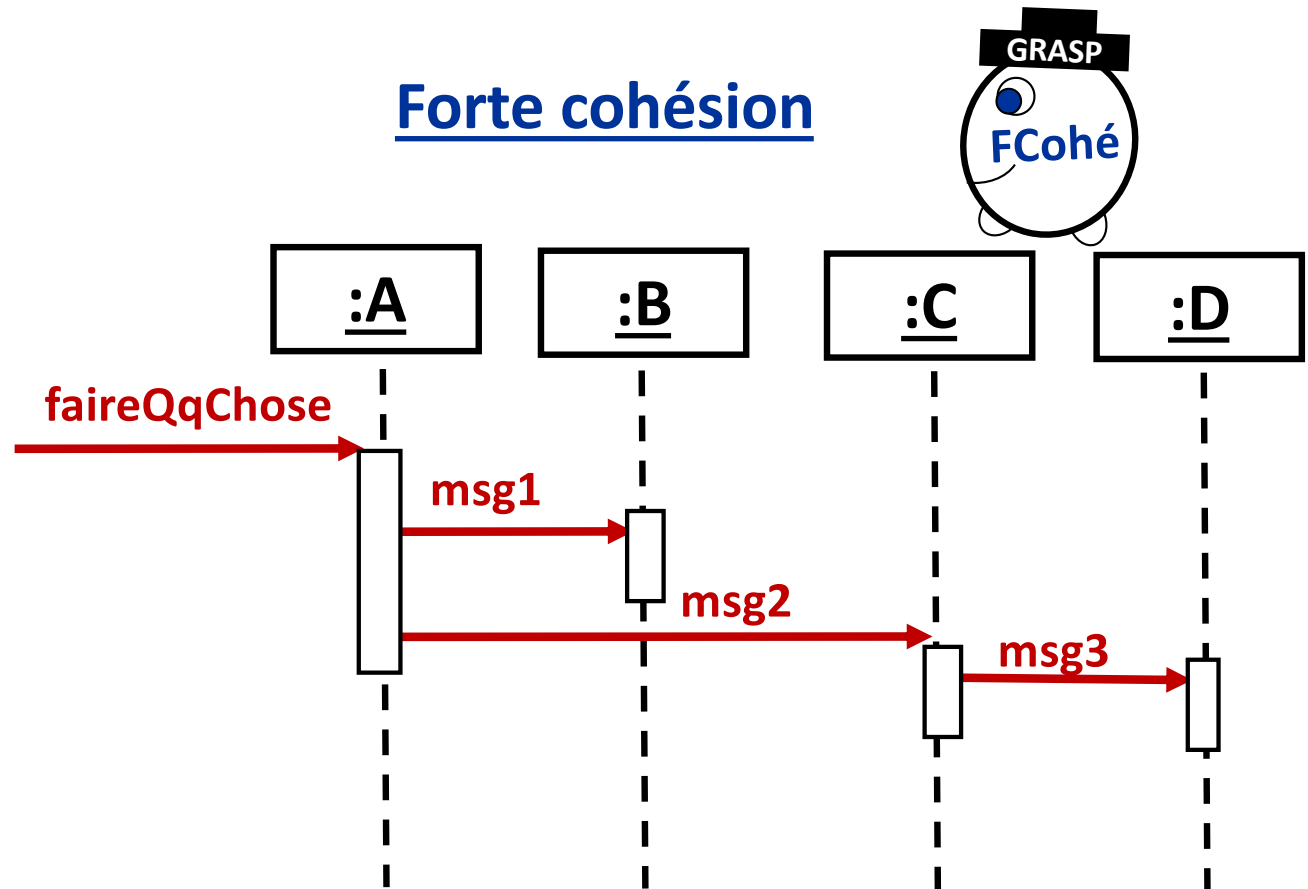


Classe « gonflée »

Centralisation

Tâches trop diversifiées

Forte cohésion



Classes «spécialisées»

Délégation et répartition du travail

Chaque classe a sa spécialité

Le comble du manque de cohésion : "La God class"



Un antipattern bien connu!

Un autre exemple



Que pensez-vous de cette classe ?

```
class HedgeHog_And_AfricanCountry
{
    private HedgeHog _hedgeHog;
    private Nation _africanNation;

    public ulong NumberOfQuills { get { return _hedgeHog.NumberOfQuills; } }
    public int CountOfAntsEatenToday { get { return _hedgeHog.AntsEatenToday.Count(); } }

    public decimal GrossDomesticProduct { get { return _africanNation.GDP; } }
    public ulong Population { get { return _africanNation.Population; } }
}
```

Solution au manque de cohésion

- Il faut scinder la classe !!

Corollaire :

Une classe qui manque de cohésion est une classe que l'on *peut facilement découper* en plusieurs classes

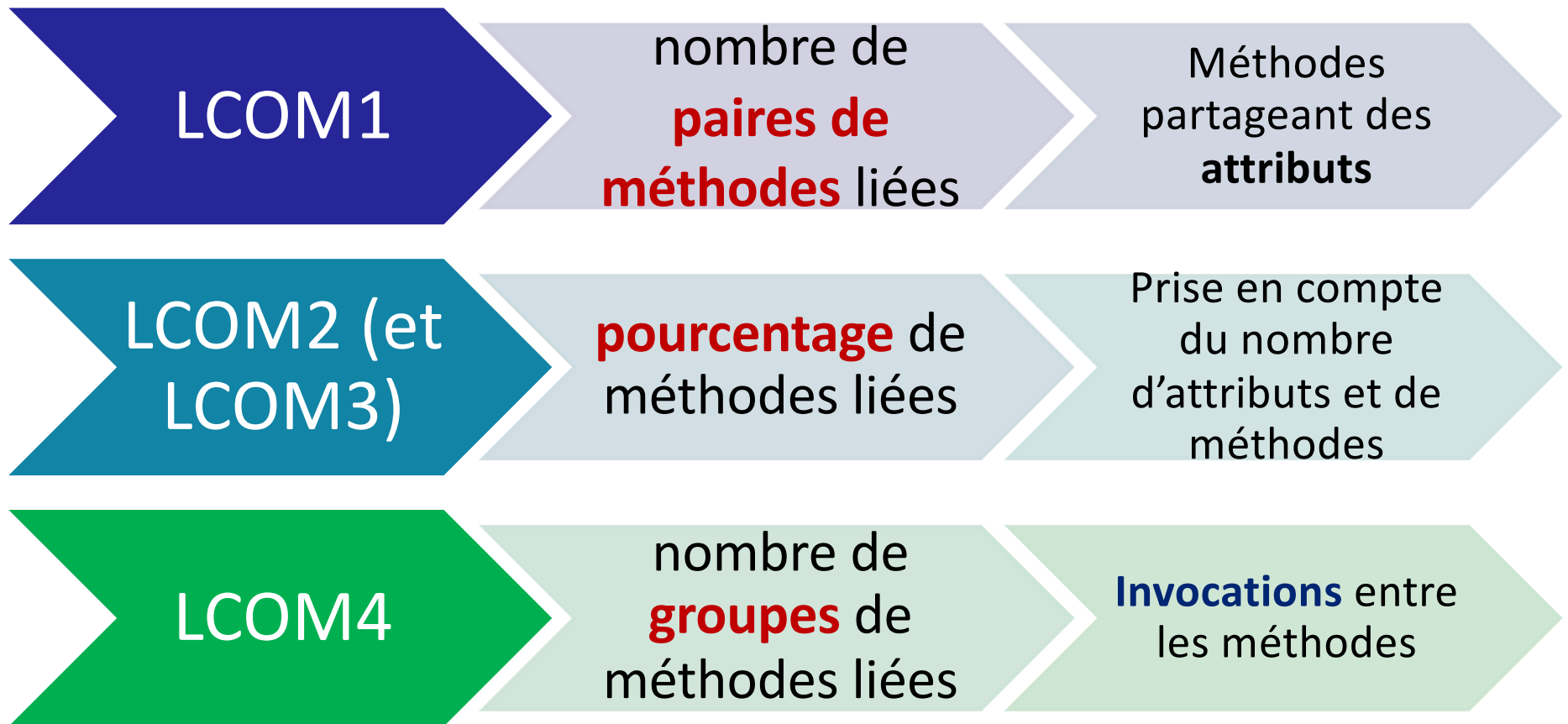
Métriques de cohésion

Comment mesurer le niveau de cohésion d'une classe?

- Compter le nombre de méthodes liées entre elles:
 - manipulant les mêmes attributs
 - s'invoquant l'une l'autre
- Comparer le nombre de méthodes liées et le nombre de méthodes indépendantes
- Identifier le nombre de groupes de méthodes liées

LCOM: Lack of Cohesion in Methods
Plusieurs variantes LCOM1, ... LCOM4

Métriques de cohésion



Métriques de cohésion

LCOM1 (Chidamber & Kemerer 1994)

- P = nombre de méthodes non liées
- Q = nombre de méthodes partageant des données

Si $P \leq Q$ LCOM1=0

Classe parfaitement cohésive

Si $P > Q$ LCOM1 = $P - Q$

Classe manquant de cohésion, elle doit être éclatée

■ Algorithme de Calcul de P et Q

- Pour chaque couple de méthode a et b , soit M_a et M_b les ensembles d'attributs manipulés respectivement dans a et b
 - Si $M_a \cap M_b = \emptyset$
 - alors $P = P + 1$ Sinon $Q = Q + 1$

LCOM compte des nombres de paires de méthodes qui accèdent ou non aux mêmes attributs

Métriques de cohésion

LCOM1 (Chidamber & Kemerer 1994)

Critique

- LCOM1=0 pour des classes de nature très différentes
- Plus LCOM est élevée moins la classe est cohésive
- Uniquement basée sur les interactions attributs-méthodes:
 - mesure limitée de la cohésion
- Cohésion sous-évaluée pour les classes qui manipulent les attributs par get/set et non directement : LCOM1 faussement élevé

Exercice1: LCOM

Calculez la métrique LCOM1 relative à la classe java TestLcom.


Démarche

P=nombre de méthodes qui ne partagent aucun attribut

Q=nombre de méthodes partageant des données

Pour chaque couple de méthode m1-m2, puis m1-m3 ...
ajouter 1 à P ou à Q selon que les méthodes manipulent ou non les ou le même attribut.

```
public class TestLcom {  
    private int a;  
    private int b;  
    public TestLcom() {  
        b=3;a=0;  
    }  
    public void m1(){  
        b=2;  
    }  
    public void m2(){  
        a=1;  
    }  
    public void m3(){  
        a=5;  
    }  
    public void m4(){  
        m5();  
    }  
    public void m5(){  
        System.out.println("OK");  
    }  
}
```



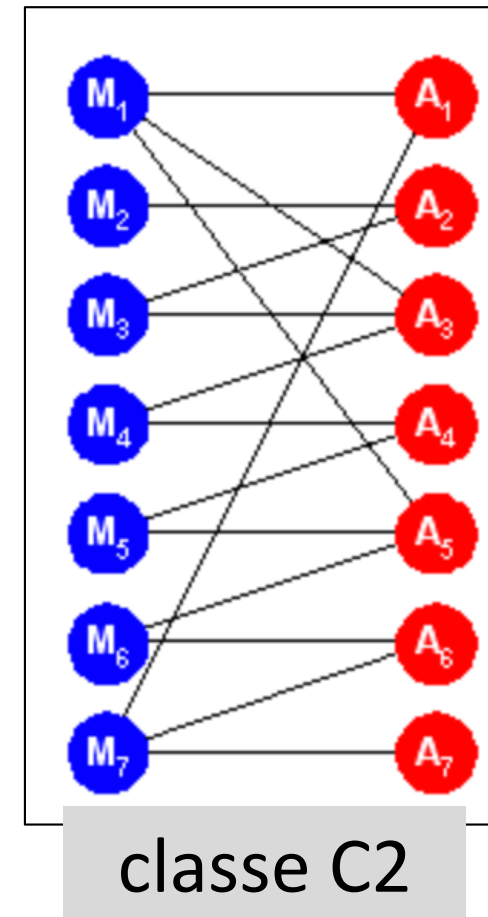
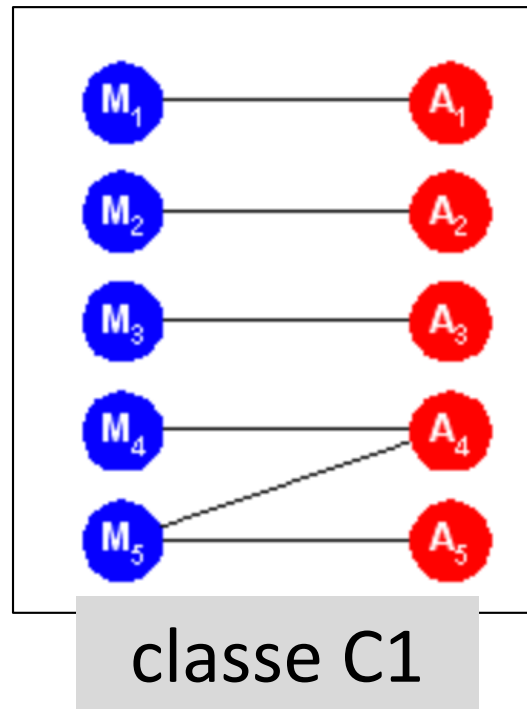
Limites LCOM



Intuitivement, quelle classe vous paraît la plus cohésive ?

Calculez LCOM1 pour vérifier.

Conclusion?



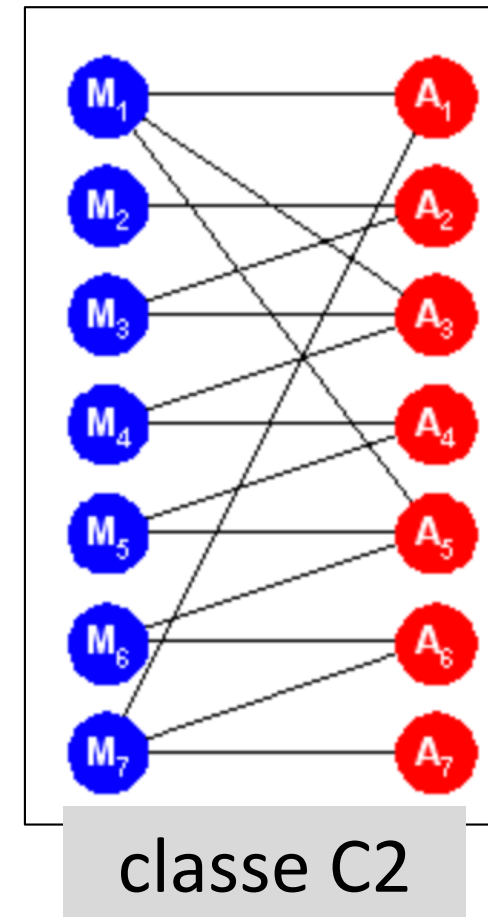
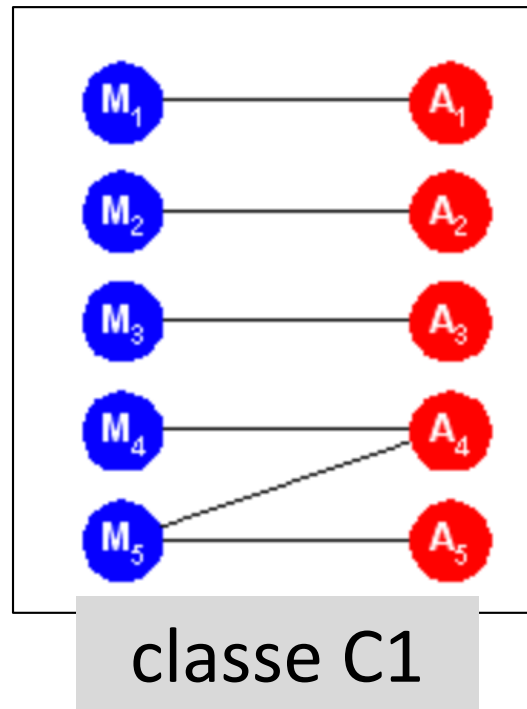
Limites LCOM

LCOM1
identiques pour
des classes très
différentes !



$LCOM1(\text{classe } C1) = 9 - 1 = 8$

$LCOM1(\text{classe } C2) = 18 - 10 = 8$



Métriques de cohésion

Pour affiner en tenant compte du nombre de méthodes

LCOM2 (Henderson & Sellers 1996)

- m = nombre de méthodes (constructeur inclus)
- nba = nombre d'attributs
- mA = nombre de méthodes qui accèdent à l'attribut A
- $\text{sum}(mA)$ = somme des mA pour tous les attributs de la classe.

$$\text{LCOM2} = 1 - (\text{sum}(mA) / (m * nba))$$

$\text{LCOM2} = 0$ si $m = 0$ ou $nba = 0$

Métriques de cohésion

LCOM2 (Henderson & Sellers 1996)

Interprétation

- LCOM2 représente le pourcentage de méthodes non liées dans une classe
- LCOM2 compris entre 0 et 1
- LCOM2 très bas= forte cohésion
- LCOM2 proche de 1 = faible cohésion problématique

Exercice 2: LCOM2

Calculez la métrique LCOM2 relative à la classe java TestLcom.

Indications

m=nombre de méthodes


a=nombre d'attributs

Ma = nombre de méthodes qui accèdent à a

Mb=

$LCOM2 = 1 - (\text{sum}(mA)/(m*a))$

```
public class TestLcom {  
    private int a;  
    private int b;  
    public TestLcom() {  
        b=3;a=0;  
    }  
    public void m1(){  
        b=2;  
    }  
    public void m2(){  
        a=1;  
    }  
    public void m3(){  
        a=5;  
    }  
    public void m4(){  
        m5();  
    }  
    public void m5(){  
        System.out.println("OK");  
    }  
}
```



Métriques de cohésion

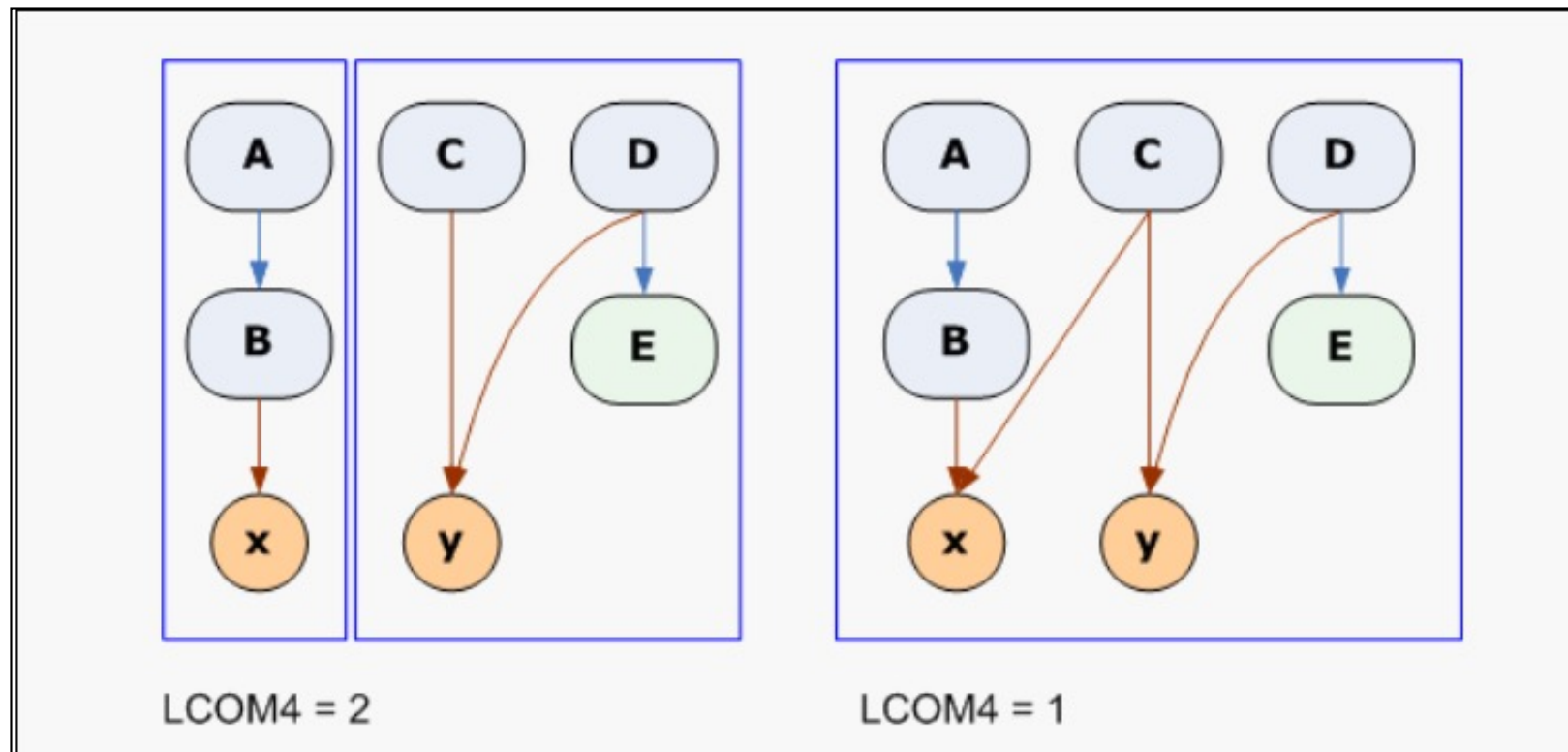
LCOM4 (Hitz et Montazeri, 1995)

- Identification des **groupes de méthodes** inter-connectées dans une classe
- Une méthode a est connectée à une méthode b si:
 - a invoque b ou b invoque a
 - a et b manipulent le même attribut
- LCOM4= nombre de groupes de méthodes inter-connectées
 - LCOM4= 1 : cohésion maximale
 - LCOM4=2: la classe doit être éclatée en deux

Métriques de cohésion

Calcul de LCOM4


Les groupes sont mis en évidence par un graphique



Exercice 3: LCOM

Calculez la métrique LCOM4 relative à la classe java TestLcom.

```
public class TestLcom {  
    private int a;  
    private int b;  
    public TestLcom() {  
        b=3;a=0;  
    }  
    public void m1(){  
        b=2;  
    }  
    public void m2(){  
        a=1;  
    }  
    public void m3(){  
        a=5;  
    }  
    public void m4(){  
        m5();  
    }  
    public void m5(){  
        System.out.println("OK");  
    }  
}
```



Exercice4 LCOM

Récupérez les classes

ContentManager, Content et DataBase sur l'ENT.

Etudiez la cohésion de la classe Content Manager.

- Approche intuitive
- Approche en s'appuyant sur des métriques

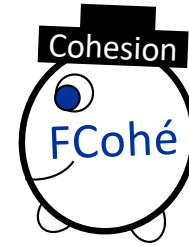
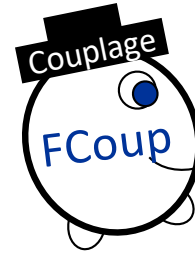


LIMITE des métriques LCOM

Que pensez-vous des classes suivantes ?

```
// This has no methods or getters, so gets a good cohesion value.  
class CustomerData  
{  
    public string FullName;  
    public Address PostalAddress;  
}  
  
// All of the getters and methods are on the same object  
class Customer  
{  
    private CustomerData _customerData;  
    public string FullName { get { return _customerData.FullName; } }  
    // etc  
}
```

	CustomerData	Customer
LCOM1	0	0
LCOM2	0	0



Liens entre Couplage et Cohésion



UNIVERSITÀ DI CORSICA
PASQUALE PAOLI

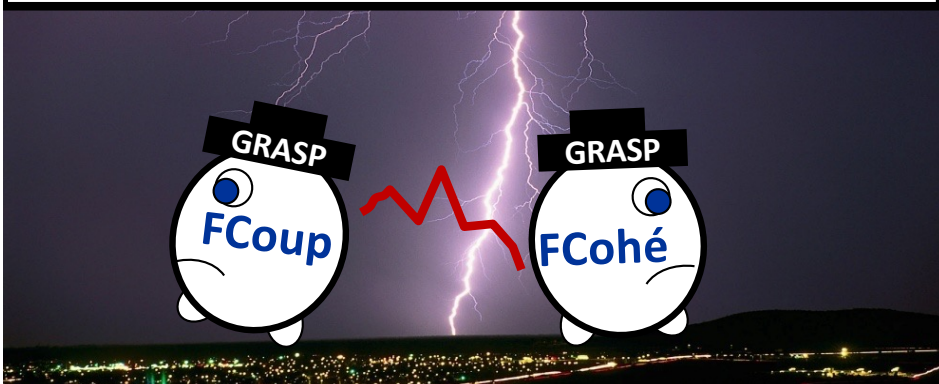
Liens Couplage/Cohésion



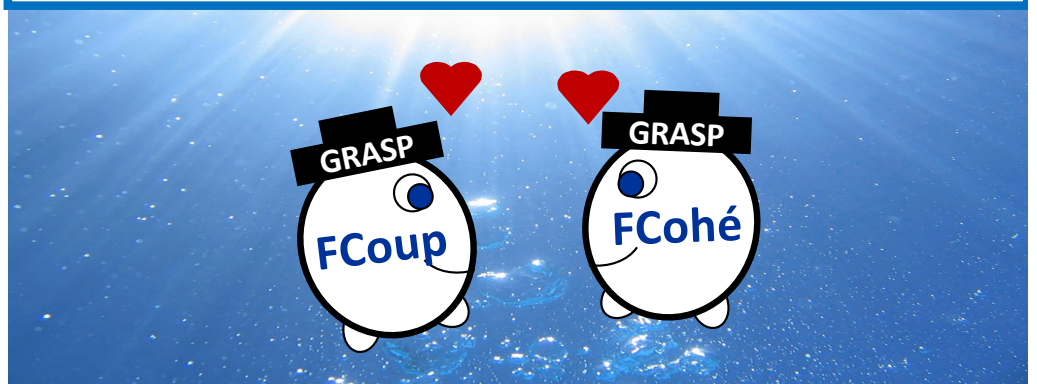
COUPLAGE/COHESION
Le Yin et le Yang du Génie
Logiciel

Couplage Faible / Forte cohésion:

Opposés en apparence



En réalité corrélés



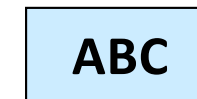
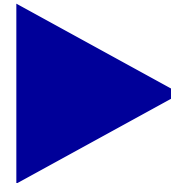
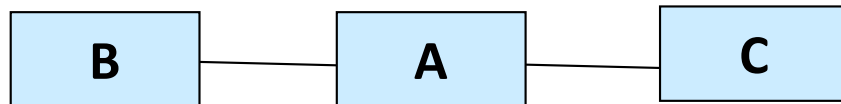
Couplage/Cohésion



Intérêts divergents

■ La Réduction du couplage peut conduire au manque de cohésion

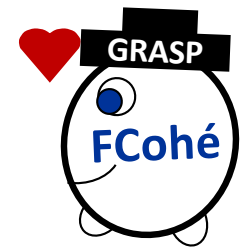
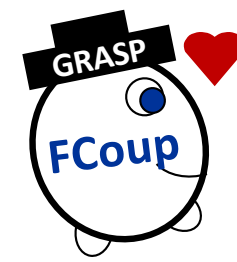
- Regroupement de classes
- Regroupement de méthodes non liées
- Diminution de la cohésion



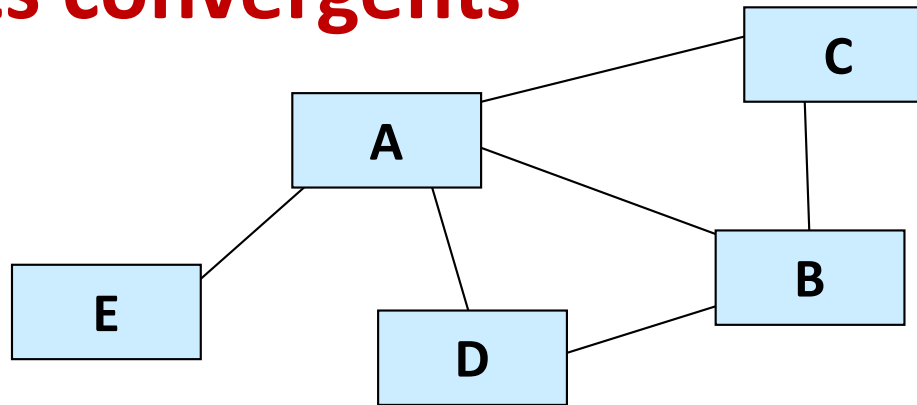
■ L'augmentation de la cohésion peut entraîner l'augmentation du couplage

- Introduction de nouvelles classes
- Multiplication des dépendances

Couplage/Cohésion

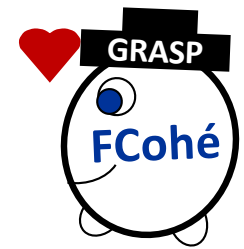
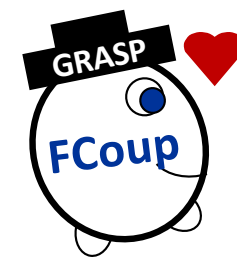


Intérêts convergents



- Un **couplage élevé** traduit souvent un manque de cohésion
 - Les classes manipulent des données localisées dans de nombreuses autres classes
 - Ces manipulations correspondent souvent à des fonctionnalités diverses sans liens entre elles
 - Faible cohésion
- Un **couplage faible** bien conçu favorise la cohésion
 - Bonne répartition des responsabilités
 - Cohésion facilitée

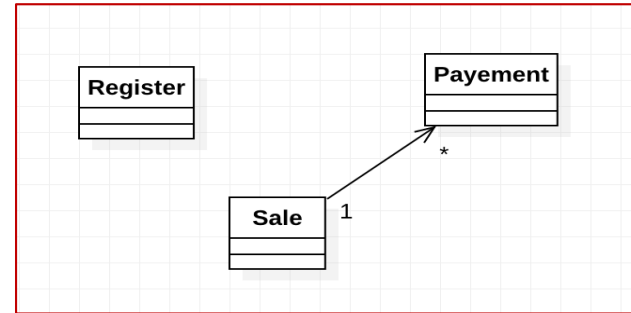
Couplage/Cohésion



Intérêts convergents

- Une **forte cohésion** facilite la diminution du couplage
 - Regroupement de fonctionnalités proches
 - Réduction du besoin de communication avec d'autres classes
 - Couplage plus faible.
- Un **manque de cohésion** augmente le risque de couplage élevé
 - les méthodes manipulent des données très différentes
 - Risque d'augmentation des besoins d'accès à des classes différentes.
 - Couplage plus fort.

Exemple Faible Couplage – Forte cohésion

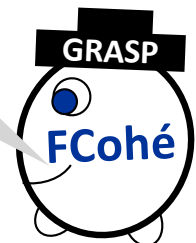
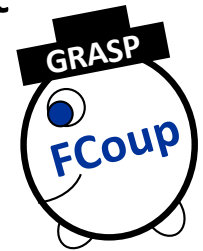


- On considère le diagramme de classe partiel suivant :
- On s'intéresse à la réalisation de l'opération `makePayment` sur un objet de la classe `Register`
- Cette opération doit créer une instance de la classe `Payement` et l'ajouter dans la liste des `Payement` d'une vente (`Sale`)
- Deux solutions proposées: comment choisir?



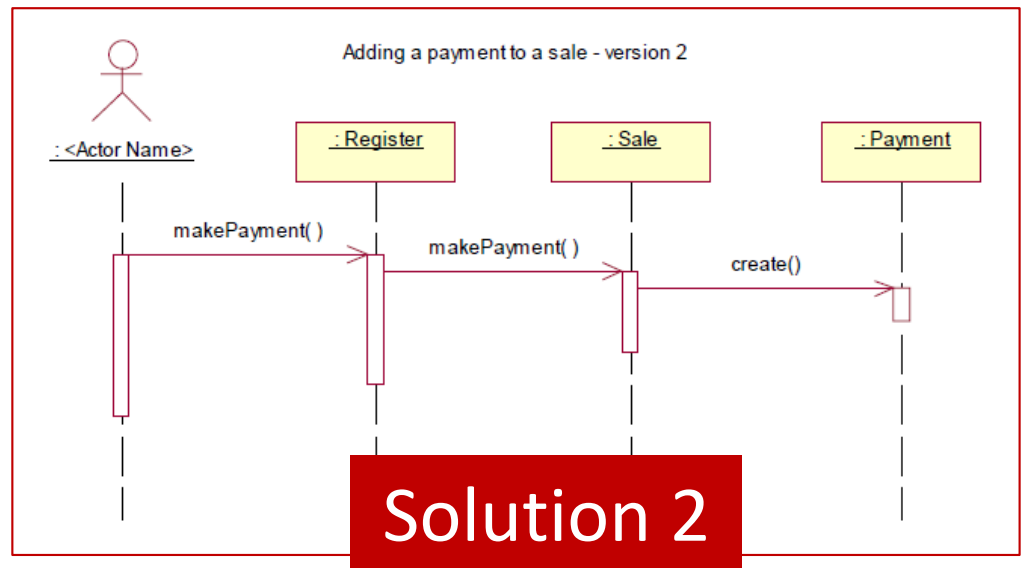
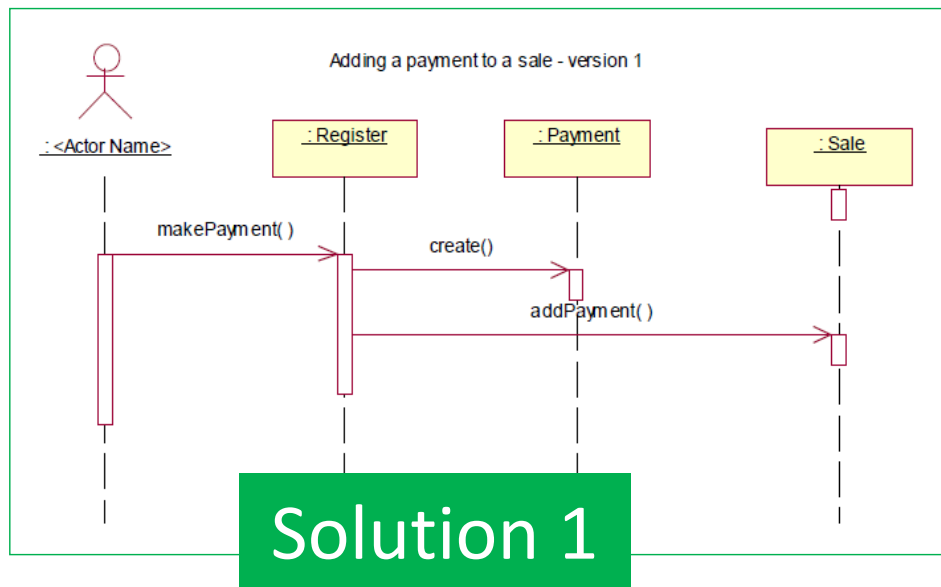
*Comment
implémenter
makePayment?*

Nous devrions
pouvoir vous
aider à choisir



Exemple Faible Couplage – Forte cohésion

1. Quelle est la meilleure solution selon le principe Faible Couplage au niveau de la classe Register?
2. Quelle est la meilleure solution selon le principe Forte Cohésion au niveau de la classe Register?



Exemple : Faible Couplage – Forte cohésion

Correction



1. Selon le principe **Faible Couplage** au niveau de la classe Register, la meilleure solution est la solution 2 car Register n'est couplée qu'à Sale alors que la solution 1 ajoute un niveau de couplage car Register devient aussi couplé avec Payment.
2. Selon le principe **Forte Cohésion** au niveau de la classe Register, la meilleure solution est la solution 2 car Register n'est responsable que de Sale alors que la solution 1 ajoute une nouvelle responsabilité à Register au niveau de la gestion du Payment.

