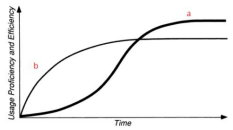
chapitre 6:

abstraction-occurrence:***contexte:***Souvent, dans le modèle du domaine, il existe un ensemble d'objets liés entre eux (*occurrences)*.Les membres d'un tel ensemble partagent une information commune.Bien qu'ils diffèrent aussi.***Problème***: Quelle est la meilleure façon de représenter un tel ensemble d'occurrences dans un diagramme de classes? ***Forces***:
Vous souhaitez représenter les propriétés de chaque ensemble d'occurrences sans avoir à dupliquer l'information commune
hiérarchie généralisée:***Contexte***:Les objets d'une hiérarchie peuvent avoir un ou plusieurs objets au-dessus d'eux (leurs supérieurs), Et un ou plusieurs objets sous eux (leurs subordonnés). Certains objets ne peuvent avoir de subordonnés ***Problème*** Comment représenter une telle hiérarchie d'objets dans laquelle certains objets ne peuvent avoir de subordonnés?***Forces Vous*** recherchez une solution flexible afin de représenter une hiérarchie générale Les objets partagent plusieurs propriétés et comportements communs
player-role:***Contexte***:Un *rôle* correspond à un ensemble de propriétés particulières associées à un objet dans un contexte particulier.Un objet peut jouer plusieurs rôles dans différents contextes.***Problème***:Comment modéliser une situation ou un objet peut jouer plusieurs rôles (conjointement ou consécutivement)?***Forces***:Il est souhaitable de renforcer l'encapsulation en intégrant l'information associée à chaque rôle dans des classes distinctes.Il faut éviter l'héritage multiple.Un objet ne peut changer de classe d'appartenance
singleton:***Contexte***:Il est fréquent de retrouver des classes pour lesquelles il ne doit exister qu'une seule instance (*singleton)****Problème***:Comment assurer qu'il ne sera jamais possible de créer plus d'une instance de cette classe?***Forces***:L'existence d'un constructeur publique ne peut garantir que plus d'une instance sera créée.L'instance du singleton doit être accessible de toutes les classes qui en ont besoin
observateur:***Contexte***:Quand une association est créée entre deux classes, celles-ci deviennent inséparables.Si vous souhaitez réutilisée l'une de ces classes, la seconde doit aussi être réutilisée.***Problème***:Comment réduire l'interconnexion entre classes, en particulier si elle appartiennent à des modules ou des sous-systèmes différents?***Forces***:Vous souhaitez maximiser la flexibilité du système
délégation:***Contexte*** Vous devez concevoir une méthode dans une classe .Vous réalisez qu'une autre classe possède une méthode qui fournit le service requis L'héritage n'est pas approprié Problème:Comment utiliser une méthode existant dans une autre classe?***Forces***:Vous souhaitez minimiser le temps de développement par la réutilisation
adaptateur:***Contexte***:Une hiérarchie existe et vous souhaitez y incorporer une classe existante.Cette classe fait aussi partie d'une autre hiérarchie.***Problème***:Comment bénéficier du polymorphisme en réutilisant une classe dont les méthodes ont les même fonctions mais pas la même signature que celles de la hiérarchie existante?***Forces***:Vous n'avez pas accès à l'héritage multiple ou vous ne voulez pas y avoir recours.
facade:***Contexte***:Souvent, une application contient plusieurs paquetages complexes.Un programmeur travaillant avec une librarie de classes doit manipuler de nombreuses classes ***Problème***:Comment simplifier la tâche des programmeurs lorsqu'ils interagissent avec une librairie complexe?***Forces***:Il est difficile pour un programmeur de comprendre et d'utiliser un sous-système entier.Si plusieurs classes d'une application appellent les méthodes de cette librairie, alors toute modification à celle-ci nécessitera une revue complète de tout le système.
immuable:***Context***:Un objet immuable est un objet dont l'état ne change jamais***Problème***: Comment créer un objet dont les instances sont immuables?***Forces***:Il ne doit exister aucun moyen d'altérer l'état d'un objet immuable***Solution***:S'assurer que le constructeur d'un objet immuable est le seul endroit ou les valeurs d'un objet sont fixées.Aucune méthode ne doit modifier l'état de l'objet.Si une méthode devrait avoir pour effet un changement d'état, alors une nouvelle instance est retournée.
mode lecture seule:***Contexte***:Il faut parfois avoir certaines classes privilégiées ayant la capacité de modifier un objet immuable***Problème***:Comment permettre une situation pèu une classe est en mode lecture seule pour certaines classes tout en étant modifiable par d'autres?***Forces***: Restreindre l'accès par les mot-clés **public**, **protected** et **private** n'est pas adéquat.
Rendre **public** une méthode, la rend accessible à tous
proxy:***Contexte***:Fréquemment, il est coûteux et complexe de créer une instance de certaines classes (ce sont des classes *lourdes*).Leur création exige du temps***Problème***:
Comment réduire la fréquence de création de classes lourdes?***Forces***:En fonction des tâches à accomplir, tous les objets d'un système doivent demeurer disponibles lors de l'exécution du système.Il est aussi important d'avoir des objets dont les valeurs persistent d'une exécution à l'autre
factory:***Contexte***:Un cadriciel réutilisable a besoin de créer des objets; toutefois, les objets à créer dépendent de l'application.***Problème***:Comment permettre à un programmeur d'ajouter des clases spécifiques à son application dans un système construit à partir d'un cadriciel? ***Forces***:Il faut que le cadriciel puisse créer des classes de l'application, bien qu'il ne connait pas ces classes.***Solution***:Le cadriciel délègue la création des classes à une classe spécialisée appelée la *Fabrique*.La fabrique est une interface générique définie dans le cadriciel.
Cette interface contient une méthode dont le but est de créer des instance de sous-classes d'une classe générique

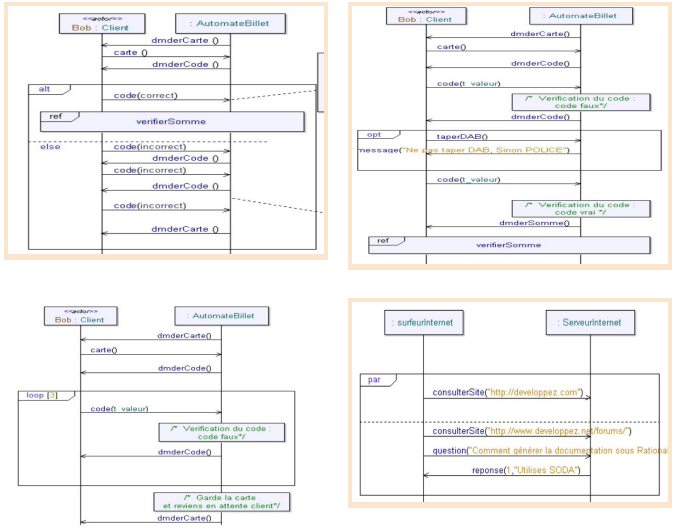
chapitre 7:

Est-ce que le système permet à l'utilisateur de réaliser ses tâches?C'est l'*utilité*
Est-ce que le système permet à l'utilisateur d'apprendre et de comprendre comment utiliser le système? C'est l'utilisabilité aussi appelée *convivialité.****L'utilisabilité comprend différents aspects:****Facilité d'apprentissage La vitesse à laquelle un nouvel utilisateur peut arriver à utiliser le système. Efficacité d'utilisation:Avec quelle rapidité un utilisateur entraîné arrive à accomplir ses tâches.Traitement des erreurs:Jusqu'à quel point le système prévient leserreurs,les détecte et aide à les corriger.Acceptation:Jusqu'à quel point les utilisateurs apprécie ce système*
evaluation heuristique:1. Choisir quelques cas-types.2. Pour chacune des fenêtres impliquées Identifier minutieusement les problèmes qui peuvent être présents,les decrire et donner une solution
evalu par observation des utilisateurs:Sélect les utilisateurs représentant les plus importants acteurs du système.select les cas-type les plus importants.Décrire des scénarios d'usage.Bien expliquer le but de l'évaluation aux utilisateurs .Filmer si possible la session.Discuter avec les utilisateurs à mesure qu'ils exécutent la tâche à faire.faire recommandation
learning curves: a =expert ; b=novice



chapitre 8:

activity:Une *activité* peut être lancée lorsque le système se trouve dans un état.Une activité a une durée.En réponse à la terminaison de l'activité, le système peut effectuer un changement d'état.Les transitions peuvent aussi se produire lorsque l'activité est en cours:L'activité se termine alors et le changement d'état s'effectue
event:
transition:Une transition représente un changement d'état en réponse à un événement.Cette transition est considérée instantanée.L'étiquette associée à une transition est l'événement causant ce changement d'état
guard condition:A condition that determines whether a certain transition
will occur in a state diagram when an event happens.
states:A tout instant, le système se trouve dans un état.Il demeura dans cet état jusqu'à l'occurrence d'un événement provoquant un changement d'état.Un état se représente à l'aide d'un rectangle arrondi contenant le nom de cet état.États spéciaux:Un disque noir représente l'état initial.Un disque noir entouré d'un cercle représente un état final
fragments:A group of messages in a sequence diagram that are treated
specially; for example they may be optional or repeated.
Represented by a box with a code in the top-left corner
indicating the kind of fragment.
concurrency:La concurrence dans un diagramme d'activité se représente à l'aide de points de rencontre, de fourchettes et de rendez-vous-.Une fourchette (*fork)* a une transition entrante et plusieurs transitions sortantes. L'exécution se sépare alors en différents fils d'exécution
-Un *rendez-vous* a plusieurs transitions entrantes et sortantes: Lorsque toutes les transitions entrante ont été effectué alors seulement peuvent être lancée les transitions sortantes
-Un point de rencontre (*join)* a de multiples transitions entrantes et une transition sortante
La transition sortante s'effectue lorsque toutes les transitions entrantes se sont produites
Chacune des transitions entrantes s'effectue dans un fil d'exécution distinct.
Lorsque qu'une transition entrante se produit, le fil correspondant est bloqué jusqu'à ce que les autres transitions soient complétées.
composite (nested states):**Un diagramme d'état peut être imbriqué dans un autre.**
Les états dans un diagramme interne sont des *sous-états*

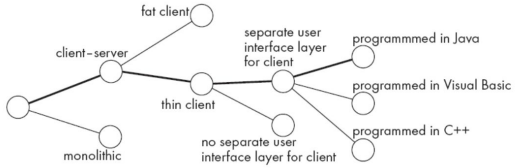


alt: soit l'utilisateur entre un code correct et dans ce cas le diagramme de séquence relatif à la vérification du code est appelé, soit (alt) l'utilisateur entre un code erroné trois fois et sa carte est gardé.
opt:L'utilisateur, si il est mécontent, peut se défouler sur le distributeur de billets. L'opérateur opt montre cette possibilité.
loop-boucle:Ce diagramme de séquence avec segment loop indique que lorsque l'utilisateur se trompe trois fois de code, la carte est gardée et le distributeur se remet en mode d'attente d'une carte.

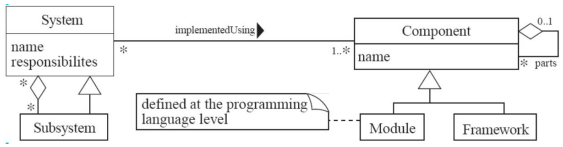
par:Un développeur averti ayant accès à Internet peut consulter en parallèle, soit le site http://www.developpez.com soit le site http://www.developpez.net/forum sans préférence d'ordre (il peut commencer par consulter les forums puis les cours, soit l'inverse)

chapitre 9: ↑COHÉSION ↓COUPLAGE

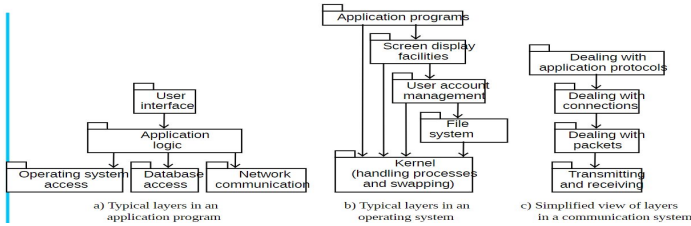
L'espace de design: ensemble des solutions possibles pour un design



Composante: élément logiciel/matériel ayant un rôle bien défini.
peut être isolée,remplacée par une autre composante(équivalentes) | devrait être réutilisable
Module: composante définie au niveau logiciel
méthodes,classes,paquetages sont des modules en Java
Système: entité logique matérielle,logicielle(les 2) ayant un ensemble de responsabilités définies implémenté avec un ensemble de module | existe toujours si composantes changées/remplacées |analyse des exigences consiste à déterminer les responsabilités d'un système.

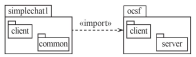


top-down: générale à specifique || éléments de bas niveau(lvl) étudiés || algorithmes & format des données. garantit une bonne structure au système
bottom-up: identifier élem de ↓ lvl || réutilisation || assemblés pour créer des structures de plus haut niveau. assure que des composantes réutilisables seront conçues.
MÉLANGE DE CES DEUX APPROCHES EST NORMALEMENT UTILISÉ!
Design: ARCHITECTURE:division en sous-système & composantes CLASSES:attributs, opérations & associations INTERFACE USAGER: ALGORITHMES:efficacité PROTOCOLES:messages et règles
Principes -> design de qualité : Accroître les profits par la réduction des coûts et l'accroissement des revenus•S'assurer de l'adhérence aux exigences•Accélérer le développement•Accroître les attributs de qualité(Utilisabilité/Efficacité/Fiabilité/Maintenabilité/Réutilisabilité) 1.Diviser pour régner Afin de maîtriser un système complexe, il faut le subdiviser en une série de plus petits systèmes•Différentes personnes peuvent ainsi travailler sur chacune des sous-parties.•Chaque ingénieur peut se spécialiser•Chacune des composantes est plus petite et plus facile à comprendre.•Certaines parties peuvent être remplacées, changées ou modifiées sans avoir à modifier les autres parties du système
2. COHÉSION:élevée si les éléments interreliés sont groupés ensemble et si les éléments indépendants sont dans des groupes distincts || plus facile à saisir et à y apporter des changements. TYPE DE COHÉSION:
Fonctionnelle: Lorsque que tout le code effectuant le calcul d'un certain résultat se trouve au même endroit ex. lorsqu'un module effectue le calcul d'un seul résultat, sans effets secondaires Bénéfices:Facilite la compréhension Plus facile à réutiliser Plus facile à remplacer. Un module gérant une base de données, créant des fichiers ou interagissant avec un utilisateur n'est pas fonctionnellement cohé
Couche: Tous les fournisseurs d'accès à un ensemble de services interreliés sont groupés ensemble. Les différentes couches devraient former une hiérarchie les couches de plus haut niveau peuvent accéder aux couches de plus bas niveaux, les couches de bas niveaux n'accèdent pas aux couches de plus haut niveau L'ensemble des procédures qu'une couche met à la disposition des autres couches pour accéder aux services qu'elle offre est l'application programming interface (API) Une couche peut être remplacée sans que cela n'affecte les autres couche il faut simplement reproduire le même API

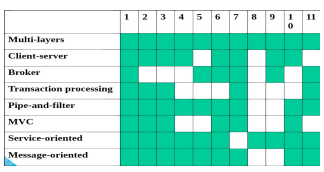


Communicationnelle: Tous les modules accédant ou manipulant les même données sont groupées ensemble
Une classe a une bonne cohésion communicationnelle si toutes les opérations de manipulation de données sont contenus dans cette classe.la classe ne gère que les données qui la concerne.Avantage: lorsqu'un changement doit être effectué sur les données, tout le code concerné se trouve au même endroit.
Séquentiel: Les procédures pour lesquelles l'une produit une sortie servant d'entrée à une autre sont groupées ensemble Ce genre de cohésion est valable lorsque les autres types de cohésion ont été achevés.
Procédurale: Les procédure qui se succèdent l'une après l'autre sont groupées ensemble Même si une ne produit pas un résultat utilisé par la suivante. •Plus faible que la cohésion séquentiel
Temporelle: Les opérations effectuées lors de la même phase d'exécution sont groupées ensemble Par exemple, tout le code utilisé lors de l'initialisation pourrait être regroupé.
Utilitaire: Tous les autres utilitaires qui ne peuvent être placés ailleurs sont groupés ensemble Un utilitaire est une procédure d'intérêt général pouvant être utilisé dans une grande variété d'applications. Les utilitaires sont hautement réutilisables Par exemple, la classe java.lang.Math

3.COUPLAGE: si interpédencance existe entre 2 modules TYPE DE COUPLAGE:
Contenue: Lorsqu'une composante subrepticement modifie les données internes d'une autre composante Le fait d'encapsuler les données réduit considérablement le couplage Elles sont déclarées private Avec des méthodes get et set
Commun: Lorsqu'une variable globale est utilisée Toutes les composantes utilisant cette variable globale deviennent alors couplées les unes aux autres Une forme plus faible de couplage est présente lorsque la variable est accessible à un nombre restreint de classes e.g. un paquetage Java Acceptable lorsque la variable globale fait référence à des paramètres globaux du système Le Singleton est un moyen d'offrir un accès contrôlé à un objet
Contrôle: Lorsqu'une procédure en appel une autre en utilisant une variable de contrôle ou une commande contrôlant l'exécution de la procédure appelée Afin d'effectuer un changement, il faut modifier à la fois l'appelé et l'appelant L'utilisation d'une opération polymorphique constitue la meilleure façon d'éviter le couplage de contrôle Une autre façon de réduire ce type de couplage consiste à avoir recours à une table look-up chaque commande est alors associée une méthode qui sera appelée lorsque ces commande est lan
Estampillage: Lorsqu'une classe est déclarée dans la liste des arguments d'une méthode Une classe en utilise donc une autre afin de réutiliser une classe, il faut aussi utiliser l'autre Pour réduire ce type de couplage utiliser une interface transmettre que des variables simples
Données: Lorsque les types des arguments sont des données simples Plus il y a d'argument, plus ce couplage est fort les méthodes appelantes doivent fournir tous ces arguments Il faut réduire ce type de couplage en évitant d'utiliser des arguments non-nécessaires Il y a souvent un compromis à faire en couplage de données et couplage d'estampillage i.e. réduire l'un accroît l'autre.
Appel: Lorsqu'une méthode en appelle une autre Ces méthodes sont couplée car le comportement de l'une dépend du comportement de l'autre Il y aura toujours du couplage d'appel dans tout système Si une même séquence d'appel se répète fréquemment, alors il faudrait songer à encapsuler cette séquence en créant une méthode regroupant ces appels.
Type: Lorsqu'un module utilise un type défini dans un autre module•Est présent chaque fois qu'une classe déclare un attribut d'une autre classe. •La conséquence est que si le type (la classe) est modifié, alors la classe qui en fait usage devra aussi être changé•Toujours utiliser le type le plus général
Inclusion: Lorsqu'une composante en importe une autre•(un paquetage en Java)ou en inclut une autre•(comme en C++). •La composante qui procède à l'inclusion devient dépendante de la composante inclue.•Si cette composante incluse est modifiée ou si quelque chose y est ajouté. il peut se produire un conflit; un nouvel élément pouvant avoir le même nom qu'un élément existant.
Externe: Lorsqu'un module dépend d'une librairie, d'un système d'exploitation, d'un matériel•Il faut réduire au maximum la dispersion de cette dépendance à travers le code.•La Façade est un moyen efficace de réduire ce type de couplage
4.Abstraction: La complexité d'un design est réduite lorsqu'un maximum de détails se trouve masqué•Une abstraction de qualité utilise toujours le principe de masquage de l'information•Une abstraction permet de saisir l'essence d'un système sans avoir à en connaître les détails de son implémentation 5.Réutilisabilité: Concevoir le design de façon à ce que les différents aspects du système soient utilisable dans différents contextes•Généraliser le design en autant que possible•Simplifier le design en autant que possible•Ajouter des options aux différents modules.La réutilisation est le principe complémentaire au principe de réutilisabilité•Réutiliser les designs existants permet de tirer profit de l'effort investi par les concepteurs de composantes réutilisables—Le clonage ne doit pas être vu comme une forme de réutilisation
.6.Flexibilité:Anticiper les changements qui seront apportés au design•Réduire le couplage et accroître la cohésion •Créer des abstractions •Ne pas introduire de constante numérique ad hoc (pas de hard-coding) •Permettre un maximum d'options—Ne pas restreindre inutilement les options•Utiliser du code réutilisable et rendre le code réutilisable 7.Portabilité:Permettre au logiciel de s'exécuter sur autant de plate-formes que possible•Éviter d'utiliser des éléments spécifiques à un environnement particulier•E.g. une librairie disponible seulement sous Microsoft Windows 8.Testabilité:Faciliter l'incorporation de tests•Concevoir un programme permettant de tester automatiquement le logiciel—Cet aspect sera discuté au chapitre 10—S'assurer que toutes les fonctionnalités du logiciel peuvent être conduites par un programme externes sans avoir à passer par l'interface utilisateur•Sous Java, il est possible de créer une méthode main()dont le rôle est de tester les autres méthodes de la classe 9.Design par contrats:Il s'agit d'une technique permettant de concevoir de façon systématique un design défensif qui demeure efficace•Principe de base—Chaque méthode conclut un contrat avec ses appelant•Ce contrat introduit un ensemble d'assertions définissant:—Quelles sont les préconditions requises pour que la méthode puisse démarrer son exécution—Quelles sont les postconditions que la méthode assure rencontrer à la fin de son exécution—Quels sont les invariants que la méthode requiert et garantie au cours de son exécution
Diagramme paquetage/communicationnel/déploiement



Architecture Multi-couches ++:Dans un système en couches, chaque couche ne communique qu'avec la couche inférieure Courtier:distribuer de façon transparente différents aspects du système Filtres:flot de données, est transformées. MVC:sépare UI. orienté Service:communique via interface.
Message:communiquent/collaborent via message
perception=utilité x utilisabilité
affordance=ensemble des opérations qu'un user peut faire à un certain instant
consistance=logiciel(produit office)



1.Diviser pour régner 2.Accroître la cohésion
3.Réduire le couplage 4.Accroître l'abstraction
5.Accroître la réutilisabilité
6.Accroître la réutilisation
7.Accroître la flexibilité
8.Anticiper l'obsolescence
9.Concevoir des designs portables
10.Faciliter les tests
11.Concevoir de façon défensive

Chapter 10:
Panne: Un comportement inacceptable ou une exécution incorrecte du système La fréquence des pannes mesure la fiabilité du système L'objectif est d'atteindre un taux d'échec très faible. Une panne résulte de la violation d'une exigence explicite ou implicite•Faute/Défaut/Bug: Une faille dans un aspect du système qui contribue ou peut contribuer à l'apparition d'une ou plusieurs pannes. Peut se trouver dans les exigences, le design ou le code Il peut prendre plusieurs défauts pour provoquer une panne particulière Aussi appelé bug•Erreur: Dans le contexte du testing, une décision inapproprié prise par un développeur qui a conduit à l'introduction d'un défaut. WHITEBOX:logique & fonctionnement interne du système BLACKBOX:sans la connaissance du code source. FLOWCHART:Basées sur l'analyse du flot de données à travers le programme.Couverture Chemin: 100%impossible Conditions:branches ayant évaluées Branches: minimale Classe d'équivalence: entrées traités de manière similaire. Bornes: élems autour des bornes.

chapitre 5:attributes, associations, system model vs domain model, reflexive associations, association classes, composition, multiplicity, operation,directionality.
Le modèle du domaine:omet plusieurs classes nécessaires à la conception du système complet Contient souvent moins de la moitié des classes du système.Devrait être indépendant: des interfaces utilisateur-des classes d'architecture•
Le modèle du système inclut::le modèle du domaine-système les classes des interfaces utilisateur les classes de l'architecture les autres classes utilitaire
classes:Représentant les types de données disponibles
associations:***Une association est utilisée afin de montrer comment deux classes sont liées entre elles***
Différents symboles sont utilisés pour indiquer la multiplicité à chaque extrémité d'une association
attributs:De simples données se trouvant dans les classes et leurs instances
opérations:Représentant les fonctions exécutées par les classes et leurs instances
généralisations:Groupant les classes en hiérarchie d'héritage

chapitre 4:
generalization,include,exclude:voir image en bas.
exigence fonctionnelle:The functional requirement is describing the behavior of the system as it relates to the system's functionality.It describes *what* a software system should do
exigence non-fonctionnelle:The non-functional requirement elaborates a performance characteristic of the system.It place constraints on *how* the system will do so.