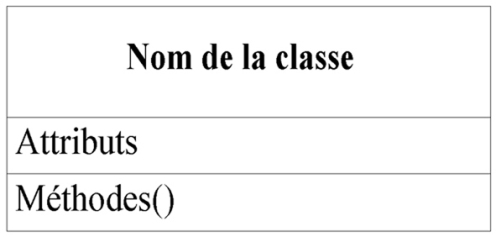
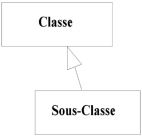
* Stéphane Crozat
* Salles
  + Cours : RB110
  + TD : FB100
* Examens
  + 25% Projet + 25% Médian + 50% Final
  + Projet
    - 10% Démonstration
    - 10% Rapport
    - 5% Soutenance
    - Analyse du projet 🡺 rapport à rendre au retour des vacances
    - Organisation : une personne pour organisation des 4 tâches :
      * Chef de projet (suivi du projet)
      * Responsable modélisation
      * Responsable technique (développement)
      * Responsable qualité
* Classes et leurs attributs :
  + <abstraite> Produits (prix, marque ou opérateur)
    - Accessoire (#modèle, stock)
    - Forfait (#nom, engagement possible ?)
    - Téléphone (OS, tactile ?, stock, #modèle)
  + <abstraite> Employés (nom, prénom, #n°, temps de travail hebdomadaire, ancienneté)
    - Equipiers
    - Gérant
  + Planning (#jour, #heure début, #heure fin)
  + Client (#n°, nom, prénom, date début)
* Associations
  + Employé — Produit — Client: Vend >
    - Attribut : date
  + Employé — Planning : Est présent >
* Méthodes
  + Employé :
    - Comment\_Prime?(): ce qu'il reste à faire pour avoir une prime
    - Planning\_personnel()
    - Prime\_montant():
    - Prime?() : oui ou non
  + *Gérant :*
    - *~~Planning\_général()~~*
    - *~~Modifier\_planning()~~*
* Base de données
  + Ensemble de données volumineux
    - Problématique d’optimisation (réponse du système assez rapide selon les besoins)
  + Données minimalement redondantes
    - Si redondance
      * 🡺 il finira par il y avoir des erreurs
      * 2 données différentes pour la même information 🡺 quelle est la bonne ?
  + Données reliées entre elles
    - Casser l’information
    - Faire des liens
  + Base de donnée centralisée : toutes les informations sont stockées dans un même endroit (contraire : distribuée)
  + Utile à plusieurs applications et utilisateurs
    - Travail parallèle possible
* SGBD
  + Logiciel
    - Structuration
    - Stockage
    - Mise à jour
    - Maintenance
  + Unique interface
    - entre les informaticiens et les données
      * Définition des schémas
      * Programmation
      * Applications
    - entre les utilisateurs et les données
      * Consultation
      * Mise à jour
  + Avant SGBD
    - Chaque application informatique nécessitait sa propre équipe de développement
    - Prend beaucoup de temps
    - Redondance
    - Problèmes de compatibilité / non portabilité
    - Fort coût
  + **Important : ce qu’apporte le SGBD**
    - Indépendance physique des données
      * On se moque de savoir comment vont être stockées les données
    - Indépendance logique
    - Manipulation par non informaticiens
    - Administration simplifiée
      * Vision globale
    - Optimisation
    - Contrôle de cohérence
      * Information ressort telle qu’elle est rentrée
    - Partageabilité
      * Données utilisables par plusieurs systèmes informatiques
    - Sécurité
    - Sûreté
      * En cas de panne logicielle ou matérielle, préservation maximum
* Donnée
  + Un truc
  + Données de données 🡺 concept flottant
* Type de données
  + Ensemble d’objets qui se ressemblent suffisamment pour qu’on les traite de la même façon
  + Exemples : les entiers, les véhicules
  + Concept tout aussi flottant 🡺 Quelle vision du monde adopter ? 🡺 Modélisation conceptuelle
* Modèles de données
  + Entité-Association a été le plus répandu
  + UML qui se généralise en informatique
    - Approche objet
    - Cf. Diagrammes de classe
  + Les deux sont à connaître
* Schéma de données
  + Décrit l’ensemble des données de la BD
  + Occurrence d’une BD : cette BD décrite au moyen de ce schéma
  + Schéma interne
    - Vu par l’informaticien, ensemble de l’instance
  + Schéma externe
    - Représentation adaptée aux besoins

Langage de données

* + Décrire et manipuler les schémas d’une BD
  + Compréhensible par l’ordinateur 🡺 on parle à l’ordinateur
  + SQL est le plus connu
  + Langage de …
    - Contrôle de Données (LCD) : définir les droits
    - Manipulation de Données (LMD) : récupérer des données, en modifier …
    - Définition de Données (LDD) : créer des données
* Administrateur
  + Responsable de la définitions des différents niveaux de schéma
* Dictionnaire de données
  + Stockage de toutes les informations relatives aux schémas et aux droits
  + C’est une base de données du SGBD lui même qui contient les données du SGBD !!!
  + « Base de données des base de données »
* Etapes de conception
  + Analyse de la situation et des besoins
    - Dialogue pour bien comprendre le problème
    - Mettre à jour le problème « très souvent le client ne demande pas ce dont il a besoin »
  + Création d’une série de modèles conceptuels
  + Modèles logiques
  + Implémentation dans un SGBD
  + Les étapes essentielles sont l’analyse et les modèles logiques
* Analyse
  + Consulter des documents
  + Consulter des experts
  + Dialoguer avec les utilisateurs à venir
    - Ceux qui vont saisir des données
    - Ceux qui vont consulter les données
    - Ceux qui vont mettre à jour les données
  + Attention aux systèmes utilisés (environnement technologique)
    - Formation des gens coûte cher 🡺 est-ce moins cher que d’utiliser ce qu’on veut
* Modèle Conceptuel de Données (MCD)
* Modèle Logique de Données (MLD)
* Modèle Physique de données (MPD)

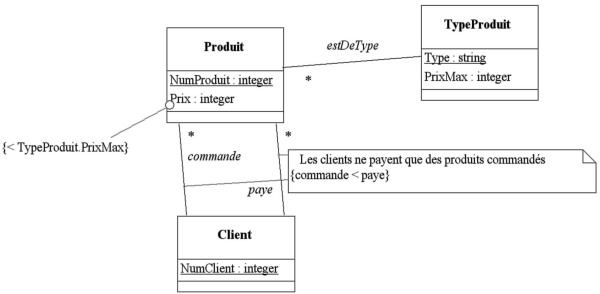
UML

* + Orienté objet
  + Fait pour « penser »
  + Langage de dessin
  + Permet de dialoguer avec le monde, les utilisateurs, car il n’est pas formel, mais est suffisamment formel pour lever les ambiguïtés 🡺 Traducteur
* Classe
  + Type abstrait
  + Propriétés communes aux objets de cette classe
  + Dessin d’une classe :
    - 
* Clé
  + Attribut ou n-uplet d’attributs
  + Identifier une instance de façon unique
  + Essentiel
* Attributs
  + Multivalué : on peut lui donner plusieurs valeurs
    - Exemple : prénoms
  + Dérivé : la valeur peut être calculée grâce à un autre attribut
    - Exemple : âge calculable grâce à la date de niassance
  + Composé : comporte des sous attributs
  + Normal
* Méthodes
  + N’existent pas en E-A
* Association
  + Précise combien d’objets de chaque classe peuvent être liés avec combien d’objets de l’autre classe (1 et un 1 seul par défaut)
  + Lien entre les objets des classes associées
* Cardinalité
  + En UML : 1 conducteur conduit plusieurs voiture : 1 du côté de la classe Conducteur, \* du côté de la classe Voiture
  + Par défaut : 1
* Héritage
  + Transmission automatique des propriétés d’une classe A à une autre A’
  + A’ hérite de A
  + 
  + Factorisation
    - Pour des classes proches, il n’est pas nécessaires de noter deux fois tous les attributs qu’elles ont en commun
  + Autre nom : association ontologique
* On ne peut pas juger un modèle sans le cahier des charges qui va avec. Le but de la représentation est de s’adapter au cahier des charges, pas de représenter la réalité.
* UML avancé

Explicitation des associations

* + Possible de préciser le sens de lecture
  + Possible de préciser le rôle des classes
* Classe d’association
  + Ajoute des propriétés à une association
  + Pas le droit de la relier à autre chose (sinon c’est une classe normale)

Association ternaire

* + Trois classes reliées par une même association (losange)
  + Ajout d’une classe d’association fréquent
  + Une association ternaire peut être réécrite avec 3 associations binaires
  + Ne pas en abuser
  + Pas de degré > 3
* Composition
  + 1..N
  + Cycles de vie liés
  + Relation forte entre une classe composite et ses classes parties
  + Losange noir du côté de la classe composite. Le 1 de la cardinalité n’est pas obligatoire.
  + Verbe non nécessaire : par défaut, « est composé de »
* Agrégation
  + Tout objet partie appartient à au moins une classe agrégat
  + Autre nom : association méréologique
* Classe abstraite
  + Une classe abstraite est une classe non instanciable. Elle exprime donc une généralisation abstraite, qui ne correspond à aucun objet existant du monde.
  + Il faut toujours qu’elle ait un héritage sinon elle ne sert à rien.
  + Notée en italique
* Contraintes
  + Contraintes de classe
  + Contraintes d’association
  + Contraintes d’attribut
  + Exemple :
* Contraintes sur les associations
  + Inclusion :
    - Si l'association inclue est instanciée, l'autre doit l'être aussi (la contrainte d'inclusion a un sens, représenté par une flèche).
  + Et :
    - Si une association est instanciée, l'autre doit l'être aussi (équivalent à une double inclusion).
  + Exclusion :
    - Les deux associations ne peuvent être instanciés en même temps.
  + Ou :
    - Au moins une des deux relations doit être instanciée.
  + XOu:
    - Exactement une des deux relations doit être instanciée.
* Package
  + Structure le schéma
* Introduit par Codd en 1970. Modèle simple et puissant.
  + Formalisme qui permet de gérer les choses indépendamment de la façon dont elles sont stockées en machine.
  + Concepts
    - Description
    - Manipulation
  + Objectifs
    - Indépendance
    - Cohérence et pas de redondance
* Domaine
  + Définition en intension
    - On explique l'ensemble des propriétés du domaine
    - Réels, entiers, …
  + Définition en extension
    - On donne l'ensemble des valeurs du domaines
    - Exemple : énuméré
* Relation
  + Toujours définie en extension
    - Ensemble des tuples qui définissent la relation
    - On peut la représenter par une table
  + Attribut d'une relation : une colonne
  + Enregistrement d'une relation : une ligne
* Clé
  + Groupe d'attributs minimum : détermine un tuple unique dans une relation
  + Toute relation contient une clé
    - Elle ne doit pas avoir deux lignes identiques
  + Il existe en général plusieurs clés dans une relation
  + Clé primaire
    - Choisie de façon arbitraire
    - La plus simple possible
  + Clé artificielle
    - Parfois, aucune clé naturelle
    - Attribut ajouté
    - Utilité : identifier de façon unique chaque enregistrement
* Schéma relationnel d'une BD
  + Définition en intension d'une BD
  + Relation (Attribut1 : Domaine1, … AttributN : DomaineN)
* Clé primaire identifiées par des # devant chaque attribut la composant
* Passage UML — R
  + 1:n : on rajoute une clé étrangère dans la relation qui a le n
  + 1:1
    - Deux choix
      * On traite cette association comme une association 1:n particulière
      * On fusionne les deux classes pour faire une seule relation
    - 1..1 : 1..1
      * On préfère souvent la fusion (plus simple)
    - 0..1 : 1..1
      * On préfère les deux relations
      * On rajoute à la classe qui a le 0..1 une clé étrangère de l'autre classe
      * Si on fusionne, c'est la clé de la classe qui a le 1..1 qui est la clé de la fusion
  + Héritage
    - Notion d'héritage exclusif : impossible d'appartenir à 2 classes qui héritent d'une même classe
    - Notion d'héritage complet ou presque complet : pas d'attributs dans les classes qui héritent, et aucune association propre
    - Trois solutions pour transformer une relation d'héritage :
      * Représenter l'héritage par une référence entre la classe mère et la classe fille
      * Représenter uniquement les classes filles par des relations
      * Représenter uniquement la classe mère par une relation
    - Par référence :
      * Classe mère non abstraite :
        + Chaque classe représentée par une relation
        + Clé primaire de la classe mère identifie les classes filles : clé primaire ET étrangère vers la classe mère !!!
        + Vue créée avec une jointure pour chaque classe fille avec la classe mère
        + Exemple :

Classe1(#a,b)

Classe2(#a=>Classe1,c,d) avec c KEY

vClasse2=jointure(Classe1,Classe2,a=a)

Classe3(#a=>Classe1,e,f) avec e KEY

vClasse3=jointure(Classe1,Classe3,a=a)

* + - * Classe mère abstraite :
        + Pareil
        + Contrainte supplémentaire :   
          PROJ(Classe1,a) IN (PROJ(Classe2,a) UNION PROJ(Classe3,a))
    - Par les classes filles :
      * Classe mère abstraite :
        + Les classes filles sont représentées
        + Les attributs de la classe mère sont répétés au niveau de chaque classe fille
        + Clé primaire de la classe mère utilisée pour identifier les classes filles
      * Classe mère non abstraite :
        + C'est pourri
        + Exemple :

Classe1(#a,b)

vClasse1=Union(Union(Classe1, Projection(Classe2, a, b)) ,Projection(Classe3, a, b))

Classe2(#a, b, c, d) avec c KEY

Classe3(#a, b, e, f) avec e KEY

Contrainte : PROJ(Classe1,a) NOT IN (PROJ(Classe2,a) UNION PROJ(Classe3,a))