POOGL

Yves Caniou

2 février 2005

Projet
UML
CVS, DIA
JAVA

Quelques rappels nécessaires sur les Projets

...Et faire tout un tas de paperasses “inutiles”

- Avant :
  - faire un cahier des charges
  - faire un échéancier
  - définir l'architecture du logiciel
  - se répartir le boulot
- Pendant
  - travailler à plusieurs
  - respecter l'échéancier
  - documenter au fur et à mesure
- Après
  - faire la doc utilisateur
  - faire une démo

Échéancier approximatif

- 2-9 février : sujets à-peu-près fixés et approuvés par moi.
  - Donc pour cela, il vaudrait peut-être m'en parler...
  - En particulier : un mail pour vendredi 4 fév donnant une présentation (provisoire et a compléter) du projet avec
    - un titre + 3 noms
    - descriptif du contexte, ce que le logiciel doit faire, etc.
    - descriptif des gdes lignes de l'architecture logicielle
    - qui fait quoi
    - de toute façon, un échéancier par mois ! (annoté et pris en compte dans la note finale)
- Février : je fais quelques cours, et vous modélisez
- Les séances du mois de mars sont consacrées à des exposés/discussions de votre modélisation
- En avril vous programmez, et je fais du support Java.
- Le code est rendu le jeudi 5 mai à 23h59 (sauf contre-ordre)
- Une journée de démos, publique, par exemple mercredi 11 mai

Échéancier approximatif

- 2-9 février : sujets à-peu-près fixés et approuvés par moi.
  - Donc pour cela, il vaudrait peut-être m'en parler...
  - En particulier : un mail pour vendredi 4 fév donnant une présentation (provisoire et a compléter) du projet avec
    - un titre + 3 noms
    - descriptif du contexte, ce que le logiciel doit faire, etc.
    - descriptif des gdes lignes de l'architecture logicielle
    - qui fait quoi
    - de toute façon, un échéancier par mois ! (annoté et pris en compte dans la note finale)
- Février : je fais quelques cours, et vous modélisez
- Les séances du mois de mars sont consacrées à des exposés/discussions de votre modélisation
- En avril vous programmez, et je fais du support Java.
- Le code est rendu le jeudi 5 mai à 23h59 (sauf contre-ordre)
- Une journée de démos, publique, par exemple mercredi 11 mai

Échéancier approximatif

- 2-9 février : sujets à-peu-près fixés et approuvés par moi.
  - Donc pour cela, il vaudrait peut-être m'en parler...
  - En particulier : un mail pour vendredi 4 fév donnant une présentation (provisoire et a compléter) du projet avec
    - un titre + 3 noms
    - descriptif du contexte, ce que le logiciel doit faire, etc.
    - descriptif des gdes lignes de l'architecture logicielle
      → les bibliothèques utilisées, paquetages, ...classes ?
    - qui fait quoi
    - de toute façon, un échéancier par mois ! (annoté et pris en compte dans la note finale)
- Février : je fais quelques cours, et vous modélisez
- Les séances du mois de mars sont consacrées à des exposés/discussions de votre modélisation
- En avril vous programmez, et je fais du support Java.
- Le code est rendu le jeudi 5 mai à 23h59 (sauf contre-ordre)
- Une journée de démos, publique, par exemple mercredi 11 mai

Échéancier approximatif

- 2-9 février : sujets à-peu-près fixés et approuvés par moi.
  - Donc pour cela, il vaudrait peut-être m'en parler...
  - En particulier : un mail pour vendredi 4 fév donnant une présentation (provisoire et a compléter) du projet avec
    - un titre + 3 noms
    - descriptif du contexte, ce que le logiciel doit faire, etc.
    - descriptif des gdes lignes de l'architecture logicielle
      → les bibliothèques utilisées, paquetages, ...classes ?
    - qui fait quoi
    - de toute façon, un échéancier par mois ! (annoté et pris en compte dans la note finale)
- Février : je fais quelques cours, et vous modélisez
- Les séances du mois de mars sont consacrées à des exposés/discussions de votre modélisation
- En avril vous programmez, et je fais du support Java.
- Le code est rendu le jeudi 5 mai à 23h59 (sauf contre-ordre)
- Une journée de démos, publique, par exemple mercredi 11 mai

Échéancier approximatif

- 2-9 février : sujets à-peu-près fixés et approuvés par moi.
  - Donc pour cela, il vaudrait peut-être m'en parler...
  - En particulier : un mail pour vendredi 4 fév donnant une présentation (provisoire et a compléter) du projet avec
    - un titre + 3 noms (un seul mail !)
    - descriptif du contexte, ce que le logiciel doit faire, etc.
    - descriptif des gdes lignes de l'architecture logicielle
      → les bibliothèques utilisées, paquetages, ...classes ?
    - qui fait quoi
    - de toute façon, un échéancier par mois ! (annoté et pris en compte dans la note finale)
- Février : je fais quelques cours, et vous modélisez
- Les séances du mois de mars sont consacrées à des exposés/discussions de votre modélisation
- En avril vous programmez, et je fais du support Java.
- Le code est rendu le jeudi 5 mai à 23h59 (sauf contre-ordre)
- Une journée de démos, publique, par exemple mercredi 11 mai

Échéancier approximatif

- 2-9 février : sujets à-peu-près fixés et approuvés par moi.
  - Donc pour cela, il vaudrait peut-être m'en parler...
  - En particulier : un mail pour vendredi 4 fév donnant une présentation (provisoire et a compléter) du projet avec
    - un titre + 3 noms (un seul mail !)
    - descriptif du contexte, ce que le logiciel doit faire, etc.
    - descriptif des gdes lignes de l'architecture logicielle
      → les bibliothèques utilisées, paquetages, ...classes ?
    - qui fait quoi
    - de toute façon, un échéancier par mois ! (annoté et pris en compte dans la note finale)
- Février : je fais quelques cours, et vous modélisez
- Les séances du mois de mars sont consacrées à des exposés/discussions de votre modélisation
- En avril vous programmez, et je fais du support Java.
- Le code est rendu le jeudi 5 mai à 23h59 (sauf contre-ordre)
- Une journée de démos, publique, par exemple mercredi 11 mai

Échéancier approximatif

- 2-9 février : sujets à-peu-près fixés et approuvés par moi.
  - Donc pour cela, il vaudrait peut-être m'en parler...
  - En particulier : un mail pour vendredi 4 fév donnant une présentation (provisoire et a compléter) du projet avec
    - un titre + 3 noms (un seul mail !)
    - descriptif du contexte, ce que le logiciel doit faire, etc.
    - descriptif des gdes lignes de l'architecture logicielle
      $\longrightarrow$ les bibliothèques utilisées, paquetages, ...classes ?
    - qui fait quoi
    - de toute façon, un échéancier par mois ! (annoté et pris en compte dans la note finale)
- Février : je fais quelques cours, et vous modélisez
- Les séances du mois de mars sont consacrées à des exposés/discussions de votre modélisation
- En avril vous programmez, et je fais du support Java.
- Le code est rendu le jeudi 5 mai à 23h59 (sauf contre-ordre)
- Une journée de démos, publique, par exemple mercredi 11 mai

Échéancier approximatif

- 2-9 février : sujets à-peu-près fixés et approuvés par moi.
  - Donc pour cela, il vaudrait peut-être m'en parler...
  - En particulier : un mail pour vendredi 4 fév donnant une présentation (provisoire et a compléter) du projet avec
    - un titre + 3 noms (un seul mail !)
    - descriptif du contexte, ce que le logiciel doit faire, etc.
    - descriptif des gdes lignes de l'architecture logicielle
      $\longrightarrow$ les bibliothèques utilisées, paquetages, ...classes ?
    - qui fait quoi
    - de toute façon, un échéancier par mois ! (annoté et pris en compte dans la note finale)
- Février : je fais quelques cours, et vous modélisez
- Les séances du mois de mars sont consacrées à des exposés/discussions de votre modélisation
- En avril vous programmez, et je fais du support Java.
- Le code est rendu le jeudi 5 mai à 23h59 (sauf contre-ordre)
- Une journée de démos, publique, par exemple mercredi 11 mai

Échéancier approximatif

- 2-9 février : sujets à-peu-près fixés et approuvés par moi.
  - Donc pour cela, il vaudrait peut-être m'en parler...
  - En particulier : un mail pour vendredi 4 fév donnant une présentation (provisoire et a compléter) du projet avec
    - un titre + 3 noms (un seul mail !)
    - descriptif du contexte, ce que le logiciel doit faire, etc.
    - descriptif des gdes lignes de l'architecture logicielle
      $\longrightarrow$ les bibliothèques utilisées, paquetages, ...classes ?
    - qui fait quoi
    - de toute façon, un échéancier par mois ! (annoté et pris en compte dans la note finale)
- Février : je fais quelques cours, et vous modélisez
- Les séances du mois de mars sont consacrées à des exposés/discussions de votre modélisation
- En avril vous programmez, et je fais du support Java.
- Le code est rendu le jeudi 5 mai à 23h59 (sauf contre-ordre)
- Une journée de démos, publique, par exemple mercredi 11 mai

Échéancier approximatif

- 2-9 février : sujets à-peu-près fixés et approuvés par moi.
  - Donc pour cela, il vaudrait peut-être m'en parler...
  - En particulier : un mail pour vendredi 4 fév donnant une présentation (provisoire et a compléter) du projet avec
    - un titre + 3 noms (un seul mail !)
    - descriptif du contexte, ce que le logiciel doit faire, etc.
    - descriptif des gdes lignes de l'architecture logicielle
      $\longrightarrow$ les bibliothèques utilisées, paquetages, ...classes ?
    - qui fait quoi
    - de toute façon, un échéancier par mois ! (annoté et pris en compte dans la note finale)
- Février : je fais quelques cours, et vous modélisez
- Les séances du mois de mars sont consacrées à des exposés/discussions de votre modélisation
- En avril vous programmez, et je fais du support Java.
- Le code est rendu le jeudi 5 mai à 23h59 (sauf contre-ordre)
- Une journée de démos, publique, par exemple mercredi 11 mai

Échéancier approximatif

- 2-9 février : sujets à-peu-près fixés et approuvés par moi.
  - Donc pour cela, il vaudrait peut-être m'en parler...
  - En particulier : un mail pour vendredi 4 fév donnant une présentation (provisoire et a compléter) du projet avec
    - un titre + 3 noms (un seul mail !)
    - descriptif du contexte, ce que le logiciel doit faire, etc.
    - descriptif des gdes lignes de l'architecture logicielle
      $\longrightarrow$ les bibliothèques utilisées, paquetages, ...classes ?
    - qui fait quoi
    - de toute façon, un échéancier par mois ! (annoté et pris en compte dans la note finale)
- Février : je fais quelques cours, et vous modélisez
- Les séances du mois de mars sont consacrées à des exposés/discussions de votre modélisation
- En avril vous programmez, et je fais du support Java.
- Le code est rendu le jeudi 5 mai à 23h59 (sauf contre-ordre)
- Une journée de démos, publique, par exemple mercredi 11 mai

Échéancier approximatif

- 2-9 février : sujets à-peu-près fixés et approuvés par moi.
  - Donc pour cela, il vaudrait peut-être m'en parler...
  - En particulier : un mail pour vendredi 4 fév donnant une présentation (provisoire et a compléter) du projet avec
    - un titre + 3 noms (un seul mail !)
    - descriptif du contexte, ce que le logiciel doit faire, etc.
    - descriptif des gdes lignes de l'architecture logicielle
      $\longrightarrow$ les bibliothèques utilisées, paquetages, ...classes ?
    - qui fait quoi
    - de toute façon, un échéancier par mois ! (annoté et pris en compte dans la note finale)
- Février : je fais quelques cours, et vous modélisez
- Les séances du mois de mars sont consacrées à des exposés/discussions de votre modélisation
- En avril vous programmez, et je fais du support Java.
- Le code est rendu le jeudi 5 mai à 23h59 (sauf contre-ordre)
- Une journée de démos, publique, par exemple mercredi 11 mai

Échéancier approximatif

- 2-9 février : sujets à-peu-près fixés et approuvés par moi.
  - Donc pour cela, il vaudrait peut-être m'en parler...
  - En particulier : un mail pour vendredi 4 fév donnant une présentation (provisoire et a compléter) du projet avec
    - un titre + 3 noms (un seul mail !)
    - descriptif du contexte, ce que le logiciel doit faire, etc.
    - descriptif des gdes lignes de l'architecture logicielle
      $\longrightarrow$ les bibliothèques utilisées, paquetages, ...classes ?
    - qui fait quoi
    - de toute façon, un échéancier par mois ! (annoté et pris en compte dans la note finale)
- Février : je fais quelques cours, et vous modélisez
- Les séances du mois de mars sont consacrées à des exposés/discussions de votre modélisation
- En avril vous programmez, et je fais du support Java.
- Le code est rendu le jeudi 5 mai à 23h59 (sauf contre-ordre)
- Une journée de démos, publique, par exemple mercredi 11 mai

Échéancier approximatif

- 2-9 février : sujets à-peu-près fixés et approuvés par moi.
  - Donc pour cela, il vaudrait peut-être m'en parler...
  - En particulier : un mail pour vendredi 4 fév donnant une présentation (provisoire et a compléter) du projet avec
    - un titre + 3 noms (un seul mail !)
    - descriptif du contexte, ce que le logiciel doit faire, etc.
    - descriptif des gdes lignes de l'architecture logicielle
      $\longrightarrow$ les bibliothèques utilisées, paquetages, ...classes ?
    - qui fait quoi
    - de toute façon, un échéancier par mois ! (annoté et pris en compte dans la note finale)
- Février : je fais quelques cours, et vous modélisez
- Les séances du mois de mars sont consacrées à des exposés/discussions de votre modélisation
- En avril vous programmez, et je fais du support Java.
- Le code est rendu le jeudi 5 mai à 23h59 (sauf contre-ordre)
- Une journée de démos, publique, par exemple mercredi 11 mai

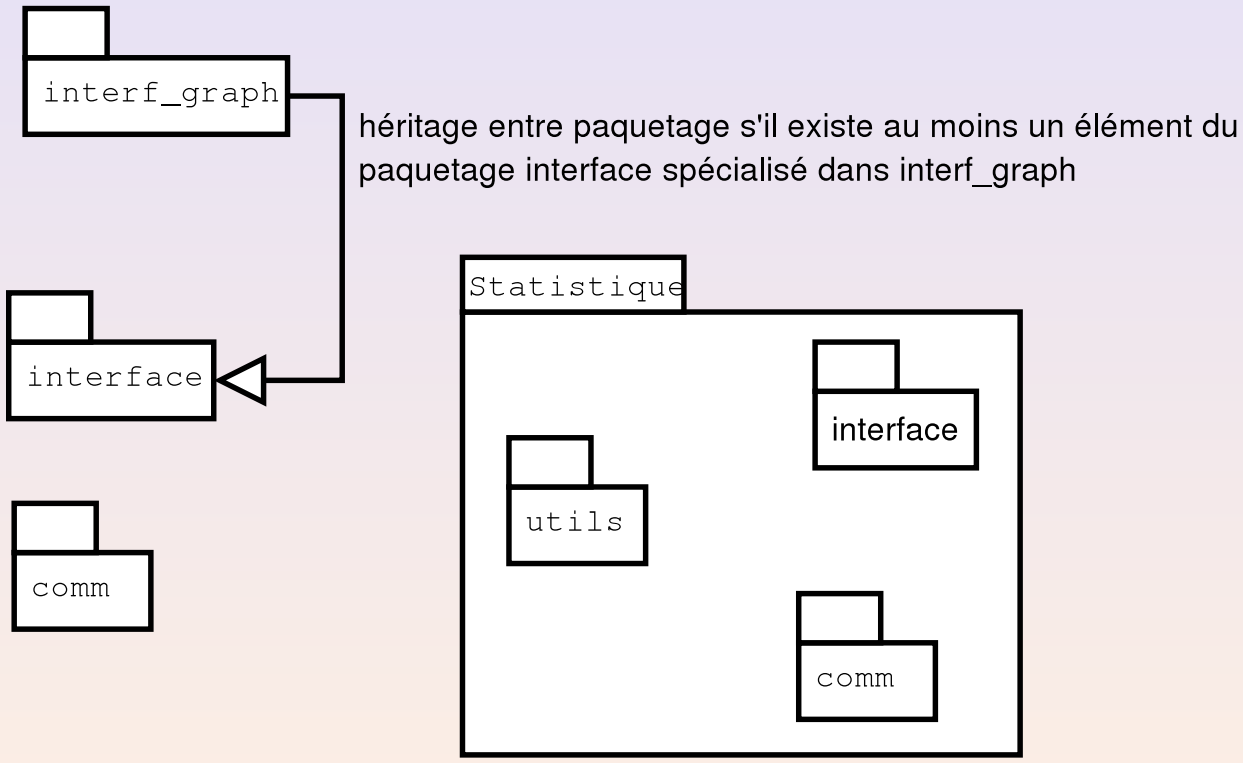
Retour sur UML

Modélisation statique d'un système

- Structurer ses modèles (paquetages, collaboration)

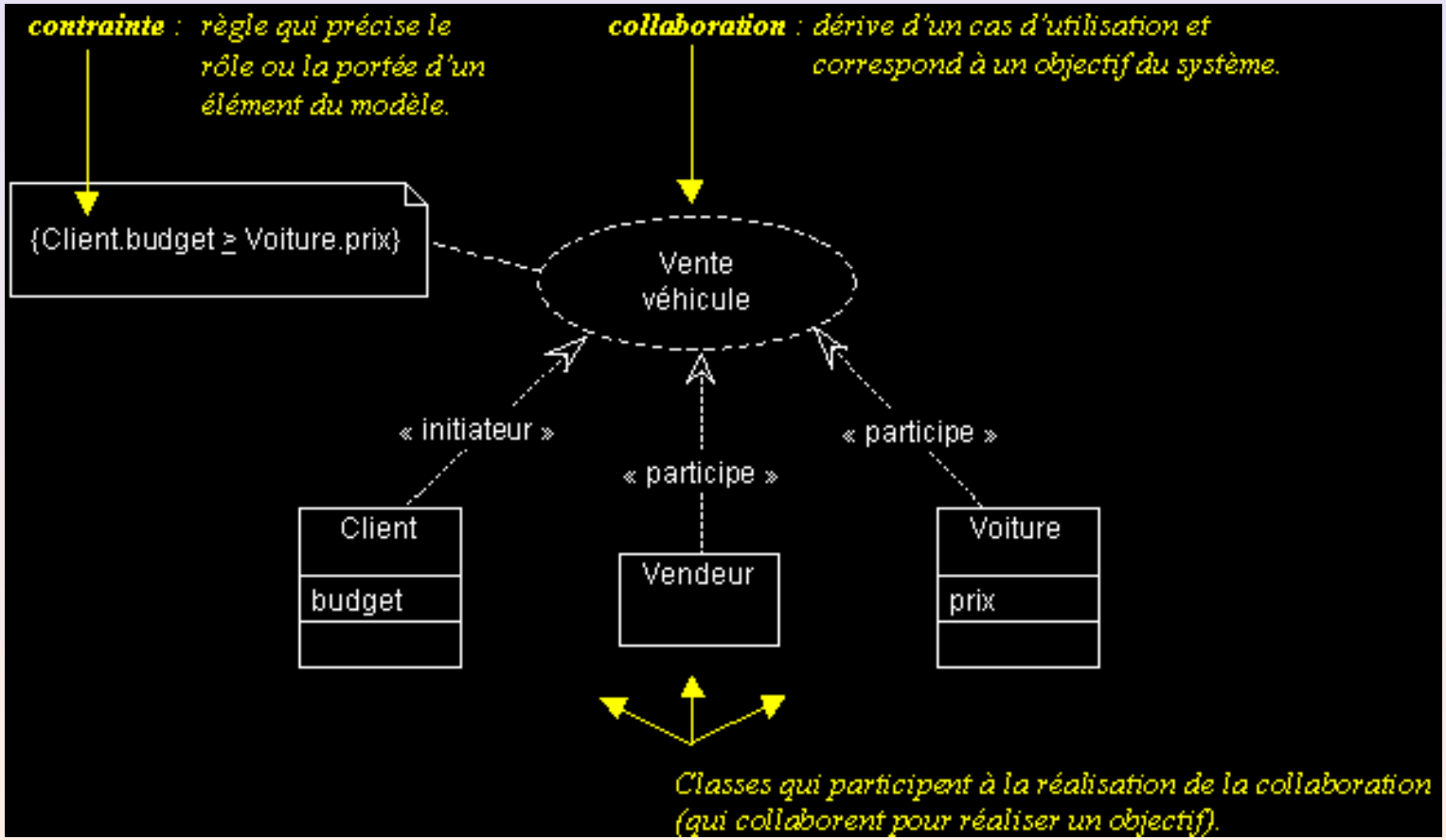
Modélisation statique d'un système

- Structurer ses modèles (paquetages, collaboration)
  Paquetages faits selon critères logiques
  $\longrightarrow$ structurer la modélisation



Modélisation statique d'un système

- Structurer ses modèles (paquetages, collaboration)
  Collaboration = interactions entre objets dans un but commun
  $\longrightarrow$ répondre à un besoin utilisateur

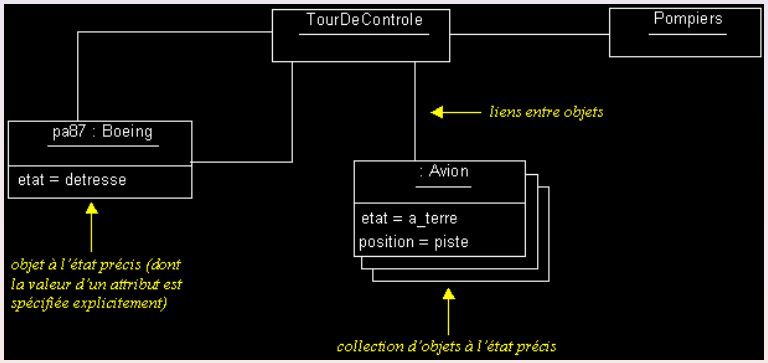


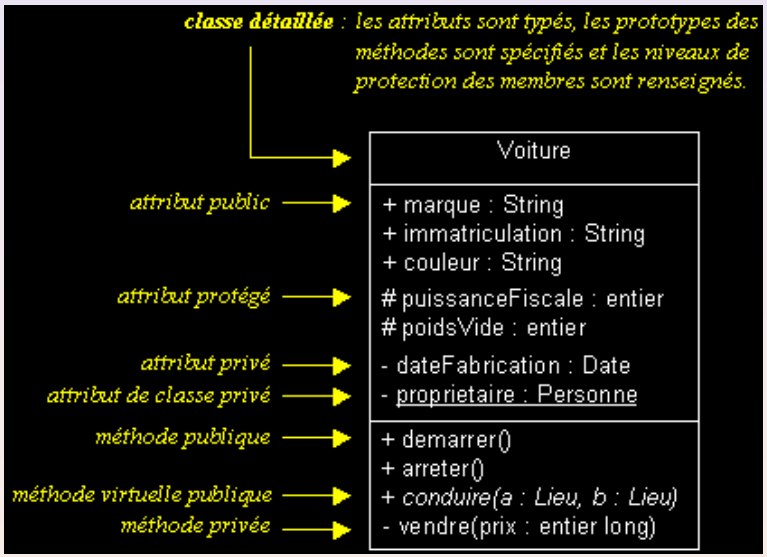
Modélisation statique d'un système

- Structurer ses modèles (paquetages, collaboration)
- Objets, diagramme d'objets et classes

Modélisation statique d'un système

- Structurer ses modèles (paquetages, collaboration)
- Objets, diagramme d'objets et classes
  Exemples :



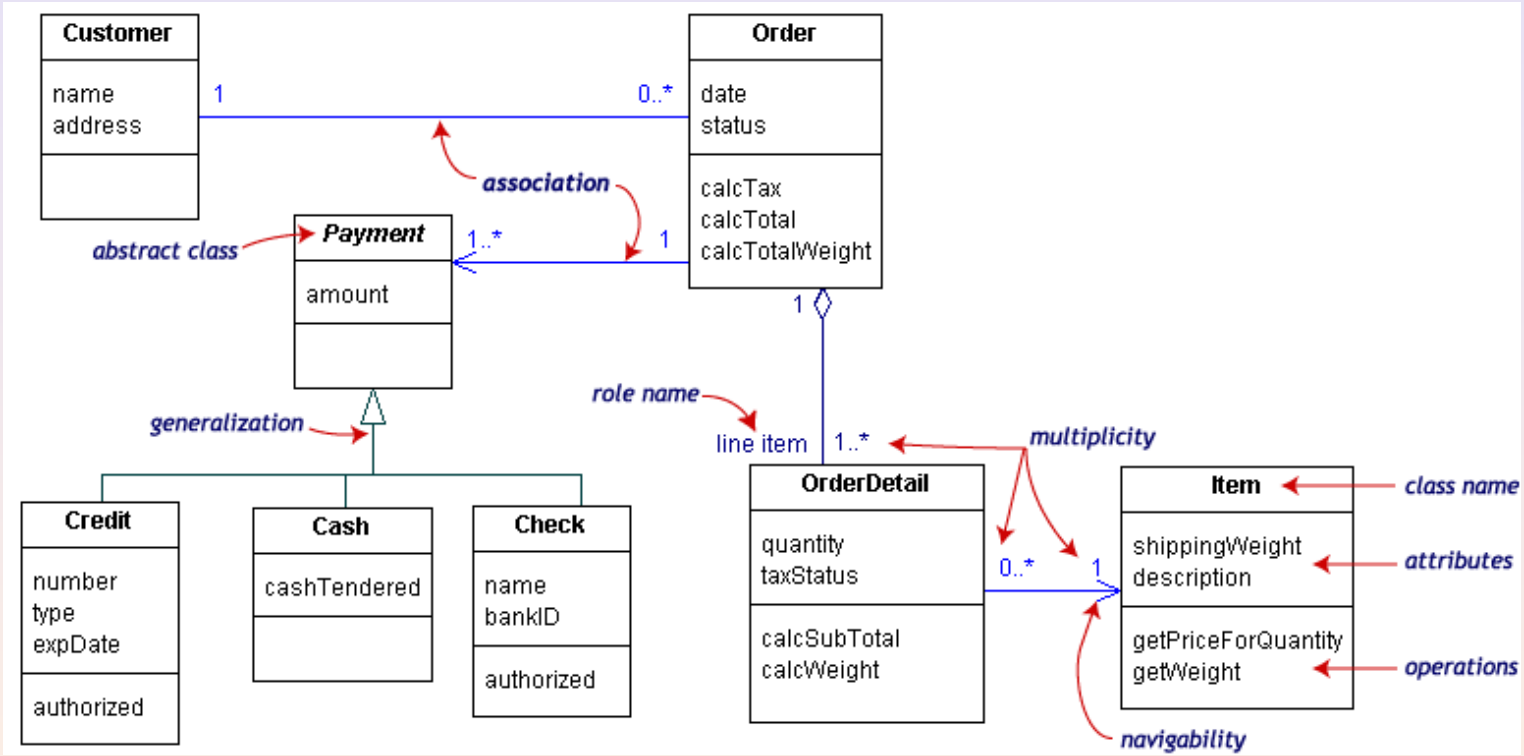


Modélisation statique d'un système

- Structurer ses modèles (paquetages, collaboration)
- Objets, diagramme d'objets et classes
- Diagramme de classes

Modélisation statique d'un système

- Structurer ses modèles (paquetages, collaboration)
- Objets, diagramme d'objets et classes
- Diagramme de classes



Modélisation statique d'un système

- Structurer ses modèles (paquetages, collaboration)
- Objets, diagramme d'objets et classes
- Diagramme de classes
- OCL

Modélisation statique d'un système

- Structurer ses modèles (paquetages, collaboration)
- Objets, diagramme d'objets et classes
- Diagramme de classes
- OCL
- Stéréotypes

Modélisation statique d'un système

- Structurer ses modèles (paquetages, collaboration)
- Objets, diagramme d'objets et classes
- Diagramme de classes
- OCL
- Stéréotypes
- Diagramme de composants

Modélisation statique d'un système

- Structurer ses modèles (paquetages, collaboration)
- Objets, diagramme d'objets et classes
- Diagramme de classes
- OCL
- Stéréotypes
- Diagramme de composants
- Diagramme de déploiement

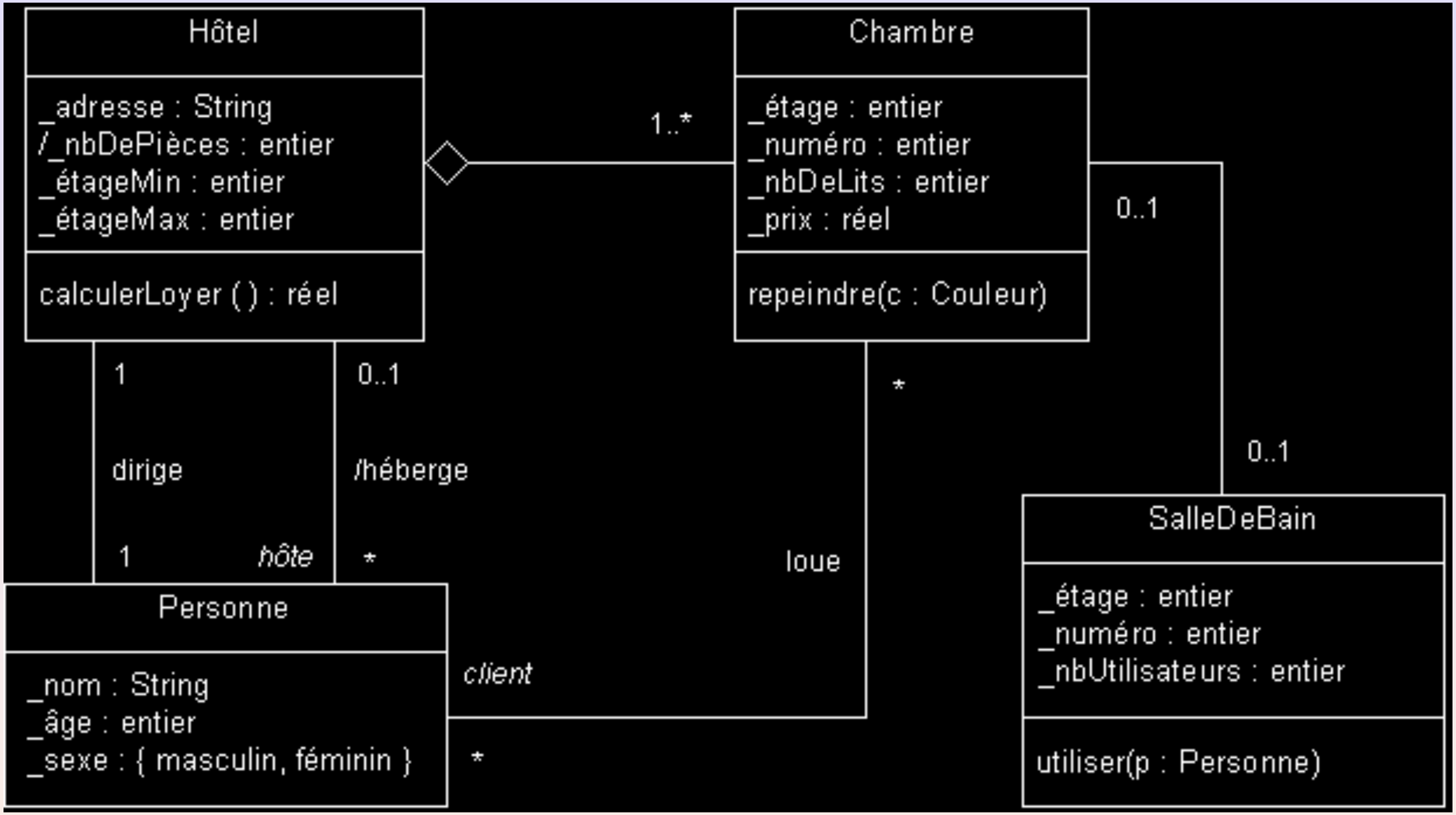
Diagramme de Classes

- Collection d'éléments de modélisation statiques qui montre la structure d'un modèle
- Ne comporte pas les aspects dynamiques et temporels du système
- Pour un modèle complexe, plusieurs diagrammes de classes complémentaires doivent être construits
- Pour représenter un contexte précis, un diagramme de classes peut être instancié en diagrammes d'objets

A titre d'exemple

Monsieur Formulain, directeur d'une chaîne d'hôtels, vous demande de concevoir une application de gestion pour ses hôtels. Voici ce que vous devez modéliser :
Un hôtel Formulain est constitué d'un certain nombre de chambres. Un responsable de l'hôtel gère la location des chambres. Chaque chambre se loue à un prix donné (suivant ses prestations).
L'accès aux salles de bain est compris dans le prix de la location d'une chambre. Certaines chambres comportent une salle de bain, mais pas toutes. Les hôtes de chambres sans salle de bain peuvent utiliser une salle de bain sur le palier. Ces dernières peuvent être utilisées par plusieurs hôtes.
Les pièces de l'hôtel qui ne sont ni des chambres, ni des salles de bain (hall d'accueil, cuisine...) ne font pas partie de l'étude (hors sujet).
Des personnes peuvent louer une ou plusieurs chambres de l'hôtel, afin d'y résider. En d'autre termes : l'hôtel héberge un certain nombre de personnes, ses hôtes (il s'agit des personnes qui louent au moins une chambre de l'hôtel...).

Diagramme résultant



Remarques

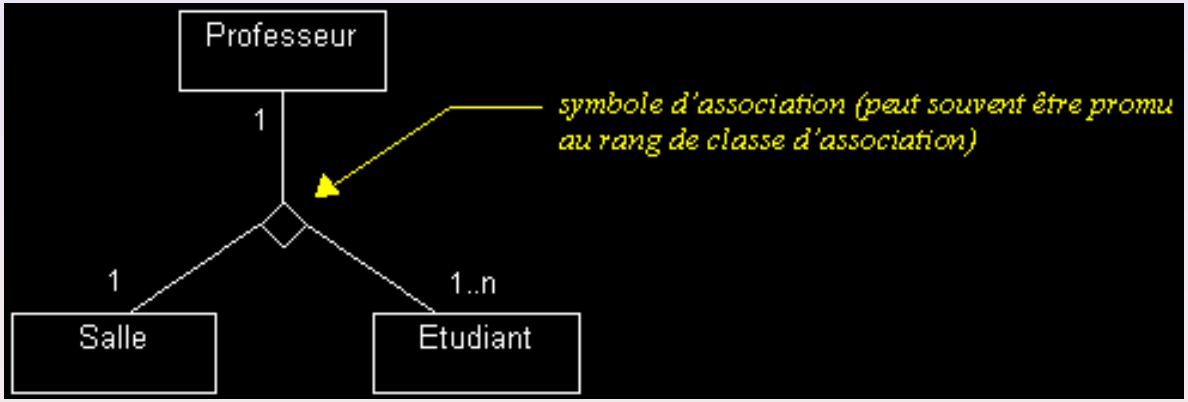
Nous n'avons vu que des relations binaires...

- Association n-aire : association qui relie plus de deux classes...

Remarques

Nous n'avons vu que des relations binaires...

- Association n-aire : association qui relie plus de deux classes...



Remarques

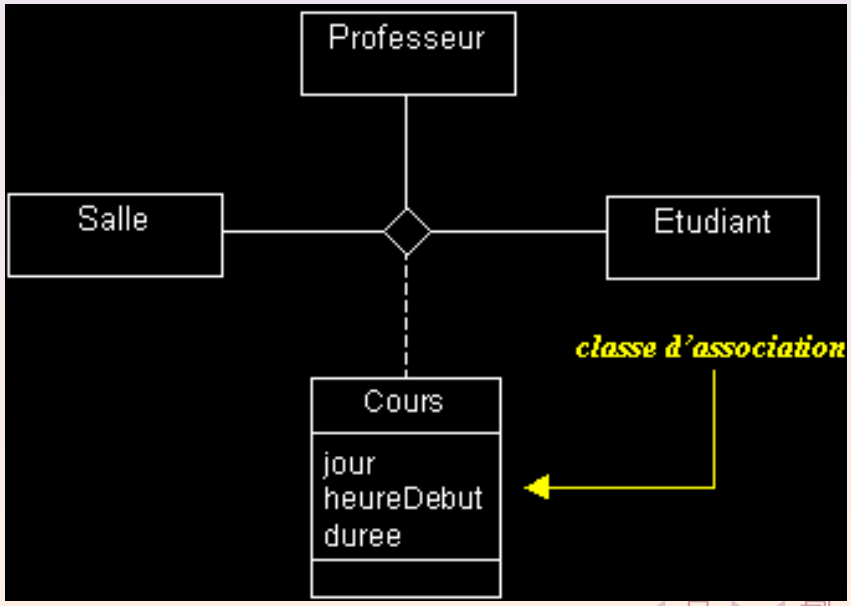
Nous n'avons vu que des relations binaires...

- Association n-aire : association qui relie plus de deux classes...
- Oui mais
  - difficiles à déchiffrer
  - peuvent enduire d'erreurs
  - donc limiter leur utilisation, en définissant de nouvelles catégories d'associations

Remarques

Nous n'avons vu que des relations binaires...

- Association n-aire : association qui relie plus de deux classes...
- Oui mais
  - difficiles à déchiffrer
  - peuvent enduire d'erreurs
  - donc limiter leur utilisation, en définissant de nouvelles catégories d'associations
- Classe d'association : classe réalisant la navigation entre les instances d'autres classes

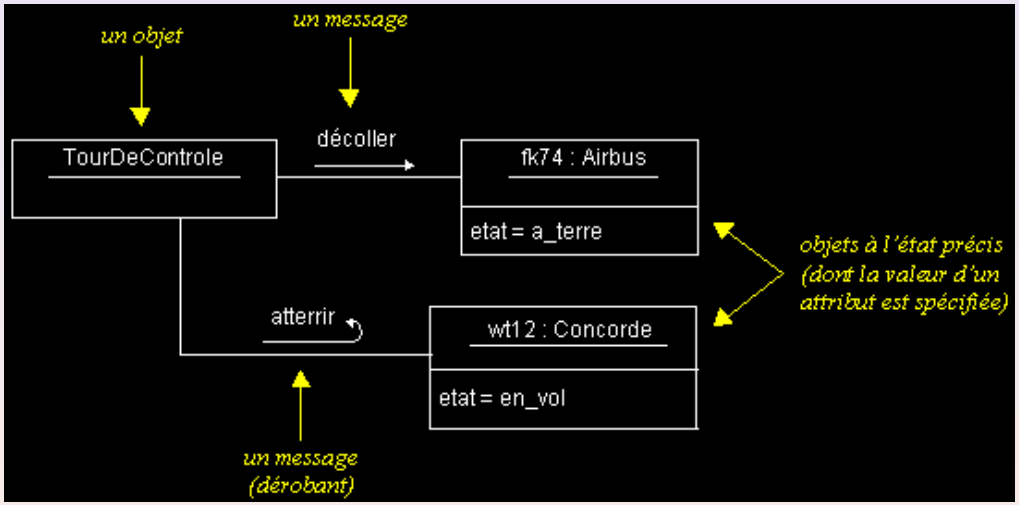


Modélisation dynamique d'un système

- Diagramme de collaboration

Modélisation dynamique d'un système

- Diagramme de collaboration
  Pour représenter le contexte d'une intéraction avec les états des objets



Modélisation dynamique d'un système

- Diagramme de collaboration
- Synchronisation des messages

Modélisation dynamique d'un système

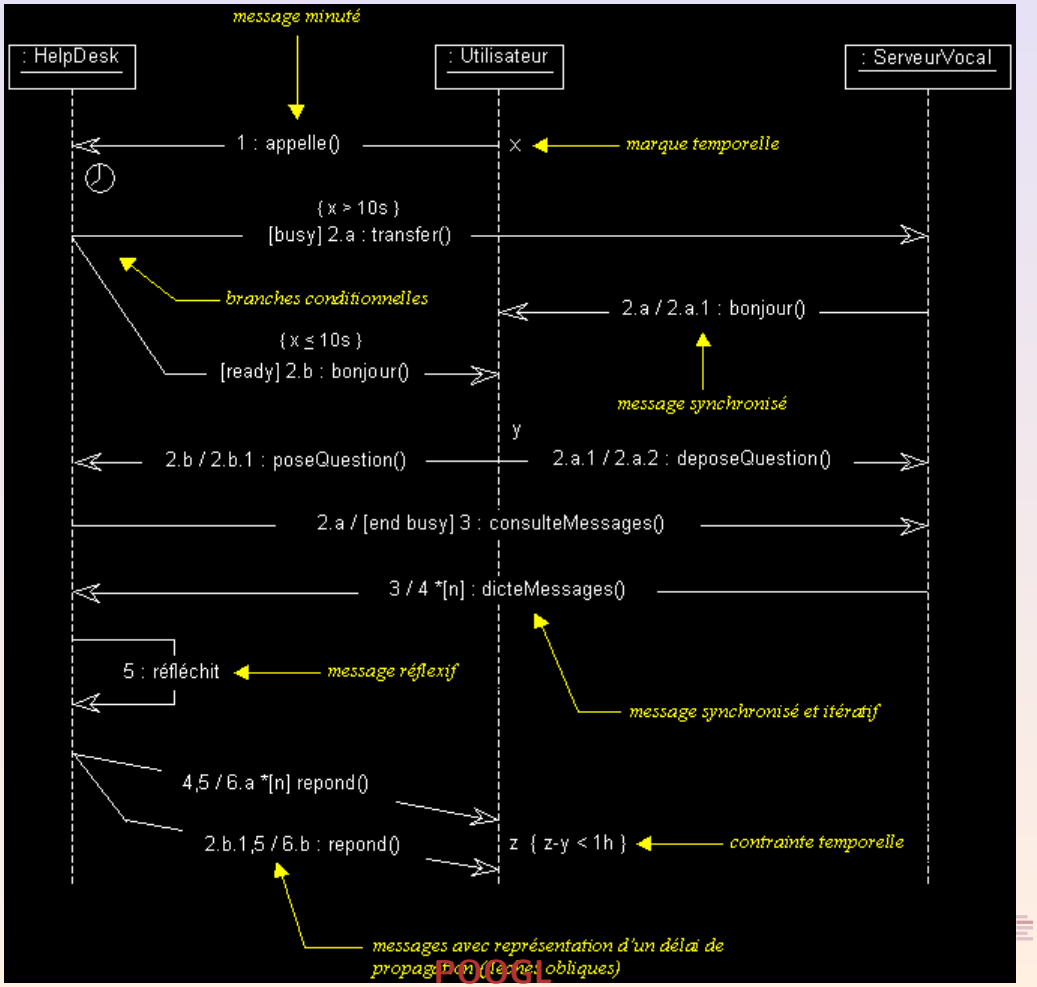
- Diagramme de collaboration
- Synchronisation des messages
- Objets actifs

Modélisation dynamique d'un système

- Diagramme de collaboration
- Synchronisation des messages
- Objets actifs
  extension des diagrammes de collaboration pour comm entre proc. ou threads

Modélisation dynamique d'un système

- Diagramme de collaboration
- Synchronisation des messages
- Objets actifs
- Diagramme de séquence



Modélisation dynamique d'un système

- Diagramme de collaboration
- Synchronisation des messages
- Objets actifs
- Diagramme de séquence
- Diagramme d'états-transitions

Modélisation dynamique d'un système

- Diagramme de collaboration
- Synchronisation des messages
- Objets actifs
- Diagramme de séquence
- Diagramme d'états-transitions
- Diagramme d'activités

CVS

CVS : Concurrent Versions System

- Objectif : permettre à plusieurs personnes de travailler sur un projet commun
  - consulter et modifier les sources
  - à distance
  - "simultanément"
  - en conservant un historique des changements effectués
  - sans consommer "trop" de ressources supplémentaires
- `http://www.gnu.org/software/cvs/`
- Frontend graphiques : Cervisia, CVSgui (daub/MAC), LinCVS (daub/MAC/linux)
- Autres projets similaires : Subversion, etc.

CVS : Concurrent Versions System

- Objectif : permettre à plusieurs personnes de travailler sur un projet commun
  - consulter et modifier les sources
  - à distance
  - “simultanément”
  - en conservant un historique des changements effectués
  - sans consommer “trop” de ressources supplémentaires
- `http://www.gnu.org/software/cvs/`
- Frontend graphiques : Cervisia, CVSgui (daub/MAC), LinCVS (daub/MAC/linux)
- Autres projets similaires : Subversion, etc.

CVS : Concurrent Versions System

- Objectif : permettre à plusieurs personnes de travailler sur un projet commun
  - consulter et modifier les sources
  - à distance
  - "simultanément"
  - en conservant un historique des changements effectués
  - sans consommer "trop" de ressources supplémentaires
- `http://www.gnu.org/software/cvs/`
- Frontend graphiques : Cervisia, CVSgui (daub/MAC), LinCVS (daub/MAC/linux)
- Autres projets similaires : Subversion, etc.

CVS : Concurrent Versions System

- Objectif : permettre à plusieurs personnes de travailler sur un projet commun
  - consulter et modifier les sources
  - à distance
  - “simultanément”
  - en conservant un historique des changements effectués
  - sans consommer “trop” de ressources supplémentaires
- `http ://www.gnu.org/software/cvs/`
- Frontend graphiques : Cervisia, CVSgui (daub/MAC), LinCVS (daub/MAC/linux)
- Autres projets similaires : Subversion, etc.

CVS : Concurrent Versions System

- Objectif : permettre à plusieurs personnes de travailler sur un projet commun
  - consulter et modifier les sources
  - à distance
  - “simultanément”
  - en conservant un historique des changements effectués
  - sans consommer “trop” de ressources supplémentaires
- `http ://www.gnu.org/software/cvs/`
- Frontend graphiques : Cervisia, CVSgui (daub/MAC), LinCVS (daub/MAC/linux)
- Autres projets similaires : Subversion, etc.

CVS : Concurrent Versions System

- Objectif : permettre à plusieurs personnes de travailler sur un projet commun
  - consulter et modifier les sources
  - à distance
  - “simultanément”
  - en conservant un historique des changements effectués
  - sans consommer “trop” de ressources supplémentaires
- `http ://www.gnu.org/software/cvs/`
- Frontend graphiques : Cervisia, CVSgui (daub/MAC), LinCVS (daub/MAC/linux)
- Autres projets similaires : Subversion, etc.

CVS : Concurrent Versions System

- Objectif : permettre à plusieurs personnes de travailler sur un projet commun
  - consulter et modifier les sources
  - à distance
  - “simultanément”
  - en conservant un historique des changements effectués
  - sans consommer “trop” de ressources supplémentaires
- `http ://www.gnu.org/software/cvs/`
- Frontend graphiques : Cervisia, CVSgui (daub/MAC), LinCVS (daub/MAC/linux)
- Autres projets similaires : Subversion, etc.

CVS : Concurrent Versions System

- Objectif : permettre à plusieurs personnes de travailler sur un projet commun
  - consulter et modifier les sources
  - à distance
  - “simultanément”
  - en conservant un historique des changements effectués
  - sans consommer “trop” de ressources supplémentaires
- `http ://www.gnu.org/software/cvs/`
- Frontend graphiques : Cervisia, CVSgui (daub/MAC), LinCVS (daub/MAC/linux)
- Autres projets similaires : Subversion, etc.

CVS : Concurrent Versions System

- Objectif : permettre à plusieurs personnes de travailler sur un projet commun
  - consulter et modifier les sources
  - à distance
  - “simultanément”
  - en conservant un historique des changements effectués
  - sans consommer “trop” de ressources supplémentaires
- `http ://www.gnu.org/software/cvs/`
- Frontend graphiques : Cervisia, CVSgui (daub/MAC), LinCVS (daub/MAC/linux)
- Autres projets similaires : Subversion, etc.

Principes

- Ne stocke que les différences entre les versions
- Permet de gérer plusieurs branches
- Chaque développeur travaille dans son répertoire
- CVS assemble les parties de chacun

Principes

- Ne stocke que les différences entre les versions
- Permet de gérer plusieurs branches
- Chaque développeur travaille dans son répertoire
- CVS assemble les parties de chacun

Principes

- Ne stocke que les différences entre les versions
- Permet de gérer plusieurs branches
- Chaque développeur travaille dans son répertoire
- CVS assemble les parties de chacun

Principes

- Ne stocke que les différences entre les versions
- Permet de gérer plusieurs branches
- Chaque développeur travaille dans son répertoire
- CVS assemble les parties de chacun

Principes

- Ne stocke que les différences entre les versions
- Permet de gérer plusieurs branches
- Chaque développeur travaille dans son répertoire
- CVS assemble les parties de chacun

Pour cela

- Tous les fichiers d'un projet sont stockés de façon centralisée dans un répertoire
  $\longrightarrow$ organiser les fichiers
- Accessibilité du répertoire sur le réseau

Principes

- Ne stocke que les différences entre les versions
- Permet de gérer plusieurs branches
- Chaque développeur travaille dans son répertoire
- CVS assemble les parties de chacun

Pour cela

- Tous les fichiers d'un projet sont stockés de façon centralisée dans un répertoire
  $\longrightarrow$ organiser les fichiers
- Accessibilité du répertoire sur le réseau

Au commencement

- Configurer les variables d'environnement
  - Dans ~/.bashrc
  - export CVS_RSH=ssh
  - export CVSROOT="login@hostname :CVS_directory"
- Si from Scratch
  - Créer le répertoire $CVROOT/easy/new
    [illegible]
    $> cvs import -m 'msg' $CVSROOT/easy/new vendortag realeasetag
    Doit être suivi d'un cvs checkout puis d'un rm !
  - Ou aussi
    $> cvs add name ; cd name ; cvs add * ; cvs commit
    Mais 'cvs add' pas récursif, préférer 'import'
- Si le projet CVS_directory/devel/name existe déjà, on le récupère
  $> cvs checkout devel/name

Au commencement

- Configurer les variables d'environnement
  - Dans ~/.bashrc
  - export CVS_RSH=ssh
  - export CVSROOT="login@hostname :CVS_directory"
- Si from Scratch
  - Créer le répertoire $CVSROOT/easy/new

    $> cvs import -m "msg" $CVSROOT/easy/new vendortag realeasetag

    Doit être suivi d'un cvs checkout puis d'un rm !
  - Ou aussi

    $> cvs add name ; cd name ; cvs add * ; cvs commit

    Mais 'cvs add' pas récursif, préférer 'import'
- Si le projet CVS_directory/devel/name existe déjà, on le récupère

  $> cvs checkout devel/name

Au commencement

- Configurer les variables d'environnement
  - Dans ~/.bashrc
  - export CVS_RSH=ssh
  - export CVSROOT="login@hostname :CVS_directory"
- Si from Scratch
  -

    $> mkdir name ; cd name ; (plein de modifs dans name);
    $> cvs import -m 'msg' $CVSROOT/easy/now vendortag realeasetag
    Doit être suivi d'un cvs checkout puis d'un rm !
  - Ou aussi
    $> cvs add name ; cd name ; cvs add * ; cvs commit
    Mais 'cvs add' pas récursif, préférer 'import'
- Si le projet CVS_directory/devel/name existe déjà, on le récupère
  $> cvs checkout devel/name

Au commencement

- Configurer les variables d'environnement
  - Dans $\tilde{/}$.bashrc
  - export CVS_RSH=ssh
  - export CVSROOT="login@hostname :CVS_directory"
- Si from Scratch
  - Créer le répertoire $CVROOT/easy/now
    $> mkdir name ; cd name ; (plein de modifs dans name)
    $> cvs import -m 'msg' $CVSROOT/easy/now vendortag realeasetag
    Doit être suivi d'un cvs checkout puis d'un rm !
  - Ou aussi
    $> cvs add name ; cd name ; cvs add * ; cvs commit
    Mais 'cvs add' pas récursif, préférer 'import'
- Si le projet CVS_directory/devel/name existe déjà, on le récupère
  $> cvs checkout devel/name

Au commencement

- Configurer les variables d'environnement
  - Dans $\tilde{/}$.bashrc
  - export CVS_RSH=ssh
  - export CVSROOT="login@hostname :CVS_directory"
- Si from Scratch
  - Créer le répertoire $CVROOT/easy/now
    $> mkdir name ; cd name ; (plein de modifs dans name)
    $> cvs import -m 'msg' $CVSROOT/easy/now vendortag realeasetag
    Doit être suivi d'un cvs checkout puis d'un rm !
  - Ou aussi
    $> cvs add name ; cd name ; cvs add * ; cvs commit
    Mais 'cvs add' pas récursif, préférer 'import'
- Si le projet CVS_directory/devel/name existe déjà, on le récupère
  $> cvs checkout devel/name

Au commencement

- Configurer les variables d'environnement
  - Dans $\tilde{/}$.bashrc
  - export CVS_RSH=ssh
  - export CVSROOT="login@hostname :CVS_directory"
- Si from Scratch
  - Créer le répertoire $CVROOT/easy/now
    $> mkdir name ; cd name ; (plein de modifs dans name)
    $> cvs import -m 'msg' $CVSROOT/easy/now vendortag realeasetag
    Doit être suivi d'un cvs checkout puis d'un rm !
  - Ou aussi
    $> cvs add name ; cd name ; cvs add * ; cvs commit
    Mais 'cvs add' pas récursif, préférer 'import'
- Si le projet CVS_directory/devel/name existe déjà, on le récupère
  $> cvs checkout devel/name

Au commencement

- Configurer les variables d'environnement
  - Dans $\tilde{/}$.bashrc
  - export CVS_RSH=ssh
  - export CVSROOT="login@hostname :CVS_directory"
- Si from Scratch
  - Créer le répertoire $CVROOT/easy/now
    $> mkdir name ; cd name ; (plein de modifs dans name)
    $> cvs import -m 'msg' $CVSROOT/easy/now vendortag realeasetag
    Doit être suivi d'un cvs checkout puis d'un rm !
  - Ou aussi
    $> cvs add name ; cd name ; cvs add * ; cvs commit
    Mais 'cvs add' pas récursif, préférer 'import'
- Si le projet CVS_directory/devel/name existe déjà, on le récupère
  $> cvs checkout devel/name

Au commencement

- Configurer les variables d'environnement
  - Dans $\tilde{/}$.bashrc
  - export CVS_RSH=ssh
  - export CVSROOT="login@hostname :CVS_directory"
- Si from Scratch
  - Créer le répertoire $CVROOT/easy/now
    $> mkdir name ; cd name ; (plein de modifs dans name)
    $> cvs import -m 'msg' $CVSROOT/easy/now vendortag realeasetag
    Doit être suivi d'un cvs checkout puis d'un rm !
  - Ou aussi
    $> cvs add name ; cd name ; cvs add * ; cvs commit
    Mais 'cvs add' pas récursif, préférer 'import'
- Si le projet CVS_directory/devel/name existe déjà, on le récupère
  $> cvs checkout devel/name

5 commandes pour survivre

- cvs checkout ou cvs co
- cvs add et cvs remove
- cvs update
- cvs commit

5 commandes pour survivre

- cvs checkout ou cvs co
- cvs add et cvs remove
- cvs update
- cvs commit
- man

JAVA

en presque 30 minutes

Bien débuter en java ☺

- Fichier source `Toto.java`, fichier objet `Toto.class`
- `javac Toto.java` crée `Toto.class` (le c c'est pour *compilo*)
- `java Toto` lance la méthode `main` de `Toto.class` dans la machine virtuelle.
- marche aussi avec kaffe
- Si on faisait une applet, c'est pas une méthode `main` qu'il faudrait.

Bien débuter en java ☺

- Fichier source `Toto.java`, fichier objet `Toto.class`
- `javac Toto.java` crée `Toto.class` (le c c'est pour *compilo*)
- `java Toto` lance la méthode `main` de `Toto.class` dans la machine virtuelle.
- marche aussi avec kaffe
- Si on faisait une applet, c'est pas une méthode `main` qu'il faudrait.

Bien débuter en java ☺

- Fichier source `Toto.java`, fichier objet `Toto.class`
- `javac Toto.java` crée `Toto.class` (le c c'est pour *compilo*)
- `java Toto` lance la méthode `main` de `Toto.class` dans la machine virtuelle.
- marche aussi avec kaffe
- Si on faisait une applet, c'est pas une méthode `main` qu'il faudrait.

Bien débuter en java ☺

- Fichier source `Toto.java`, fichier objet `Toto.class`
- `javac Toto.java` crée `Toto.class` (le c c'est pour *compilo*)
- `java Toto` lance la méthode `main` de `Toto.class` dans la machine virtuelle.
- marche aussi avec kaffe
- Si on faisait une applet, c'est pas une méthode `main` qu'il faudrait.

Bien débuter en java ☺

- Fichier source `Toto.java`, fichier objet `Toto.class`
- `javac Toto.java` crée `Toto.class` (le c c'est pour *compilo*)
- `java Toto` lance la méthode `main` de `Toto.class` dans la machine virtuelle.
- marche aussi avec kaffe
- Si on faisait une applet, c'est pas une méthode `main` qu'il faudrait.

Bien débuter en java ☺

- Fichier source `Toto.java`, fichier objet `Toto.class`
- `javac Toto.java` crée `Toto.class` (le c c'est pour *compilo*)
- `java Toto` lance la méthode `main` de `Toto.class` dans la machine virtuelle.
- marche aussi avec kaffe
- Si on faisait une applet, c'est pas une méthode `main` qu'il faudrait.

Traditionnel Hello World

Le code source :

```
class Hello {
  // pas d'attributs ni de méthodes, sauf
  // la méthode main, statique.
  //  cela veut dire quoi, statique, déja ?
  public static void main(String[] args) {
     System.out.println("Hello, java ?");
  }
}
```

Un peu plus loin..

```
// HelloDate.java
import java.util.*;
public class HelloDate {

   public static void main(String[] args) {
      System.out.println("Hello, it's: ");
      System.out.println(new Date());
   }
}
```

Donnera par exemple
Hello, it's :
Wed Feb 02 16 :35 :48 GMT+01 :00 2005

Les trucs reposants

- *Une classe, un fichier*
  - ⊕ Pas de fichiers `.h`
  - ⊕ La doc du code n'est plus le fichier `.h`, c'est une vraie doc crée par `javadoc`
  - ⊕ Pas besoin de `Makefile` ni d'`autoconf`
  - ⊕ Pas de bug dû au préprocesseur
  - ⊖ Si cela se trouve, c'est même mieux que Caml
- *Édition de lien dynamique*
  - Lorsque la machine virtuelle a besoin de créer un objet de classe `Toto`, elle charge `Toto.class`
    - Il faut qu'elle sache où le trouver (on verra plus loin)
  - ⊕ Pas d'éditeur de lien
  - ⊕ Pas de `Makefile` (je l'ai déjà dit ?)
  - ⊕ Les chemins pour la compilation, pour l'édition de liens et pour l'exécution sont les mêmes.
  - ⊖ Performance : des accès disques (et même réseau) cachés au milieu de votre programme

Les trucs reposants

- *Une classe, un fichier*
  - ⊕ Pas de fichiers `.h`
  - ⊕ La doc du code n'est plus le fichier `.h`, c'est une vraie doc crée par `javadoc`
  - ⊕ Pas besoin de `Makefile` ni d'`autoconf`
  - ⊕ Pas de bug dû au préprocesseur
  - ⊖ Si cela se trouve, c'est même mieux que Caml
- *Édition de lien dynamique*
  - Lorsque la machine virtuelle a besoin de créer un objet de classe `Toto`, elle charge `Toto.class`
  - Il faut qu'elle sache où le trouver (on verra plus loin)
  - ⊕ Pas d'éditeur de lien
  - ⊕ Pas de `Makefile` (je l'ai déjà dit ?)
  - ⊕ Les chemins pour la compilation, pour l'édition de liens et pour l'exécution sont les mêmes.
  - ⊖ Performance : des accès disques (et même réseau) cachés au milieu de votre programme

Les trucs reposants

- *Une classe, un fichier*
  - ⊕ Pas de fichiers `.h`
  - ⊕ La doc du code n'est plus le fichier `.h`, c'est une vraie doc crée par `javadoc`
  - ⊕ Pas besoin de `Makefile` ni d'`autoconf`
  - ⊕ Pas de bug dû au préprocesseur
  - ⊖ Si cela se trouve, c'est même mieux que Caml
- *Édition de lien dynamique*
  - Lorsque la machine virtuelle a besoin de créer un objet de classe `Toto`, elle charge `Toto.class`
    - Il faut qu'elle sache où le trouver (on verra plus loin)
  - ⊕ Pas d'éditeur de lien
  - ⊕ Pas de `Makefile` (je l'ai déjà dit ?)
  - ⊕ Les chemins pour la compilation, pour l'édition de liens et pour l'exécution sont les mêmes.
  - ⊖ Performance : des accès disques (et même réseau) cachés au milieu de votre programme

Les trucs reposants

- *Une classe, un fichier*
  - ⊕ Pas de fichiers `.h`
  - ⊕ La doc du code n'est plus le fichier `.h`, c'est une vraie doc crée par `javadoc`
  - ⊕ Pas besoin de `Makefile` ni d'`autoconf`
  - ⊕ Pas de bug dû au préprocesseur
  - ⊖ Si cela se trouve, c'est même mieux que Caml
- *Édition de lien dynamique*
  - Lorsque la machine virtuelle a besoin de créer un objet de classe `Toto`, elle charge `Toto.class`
  - Il faut qu'elle sache où le trouver (on verra plus loin)
  - ⊕ Pas d'éditeur de lien
  - ⊕ Pas de `Makefile` (je l'ai déjà dit ?)
  - ⊕ Les chemins pour la compilation, pour l'édition de liens et pour l'exécution sont les mêmes.
  - ⊖ Performance : des accès disques (et même réseau) cachés au milieu de votre programme

Les trucs reposants

- *Une classe, un fichier*
  - ⊕ Pas de fichiers `.h`
  - ⊕ La doc du code n'est plus le fichier `.h`, c'est une vraie doc crée par `javadoc`
  - ⊕ Pas besoin de `Makefile` ni d'`autoconf`
  - ⊕ Pas de bug dû au préprocesseur
  - ⊖ Si cela se trouve, c'est même mieux que Caml
- *Édition de lien dynamique*
  - Lorsque la machine virtuelle a besoin de créer un objet de classe `Toto`, elle charge `Toto.class`
  - Il faut qu'elle sache où le trouver (on verra plus loin)
  - ⊕ Pas d'éditeur de lien
  - ⊕ Pas de `Makefile` (je l'ai déjà dit ?)
  - ⊕ Les chemins pour la compilation, pour l'édition de liens et pour l'exécution sont les mêmes.
  - ⊖ Performance : des accès disques (et même réseau) cachés au milieu de votre programme

Les trucs reposants

- *Une classe, un fichier*
  - ⊕ Pas de fichiers `.h`
  - ⊕ La doc du code n'est plus le fichier `.h`, c'est une vraie doc crée par `javadoc`
  - ⊕ Pas besoin de `Makefile` ni d'`autoconf`
  - ⊕ Pas de bug dû au préprocesseur
  - ⊖ Si cela se trouve, c'est même mieux que Caml
- *Édition de lien dynamique*
  - Lorsque la machine virtuelle a besoin de créer un objet de classe `Toto`, elle charge `Toto.class`
  - Il faut qu'elle sache où le trouver (on verra plus loin)
  - ⊕ Pas d'éditeur de lien
  - ⊕ Pas de `Makefile` (je l'ai déjà dit ?)
  - ⊕ Les chemins pour la compilation, pour l'édition de liens et pour l'exécution sont les mêmes.
  - ⊖ Performance : des accès disques (et même réseau) cachés au milieu de votre programme

Les trucs reposants

- *Une classe, un fichier*
  - ⊕ Pas de fichiers `.h`
  - ⊕ La doc du code n'est plus le fichier `.h`, c'est une vraie doc crée par `javadoc`
  - ⊕ Pas besoin de `Makefile` ni d'`autoconf`
  - ⊕ Pas de bug dû au préprocesseur
  - ⊖ Si cela se trouve, c'est même mieux que Caml
- *Édition de lien dynamique*
  - Lorsque la machine virtuelle a besoin de créer un objet de classe `Toto`, elle charge `Toto.class`
  - Il faut qu'elle sache où le trouver (on verra plus loin)
  - ⊕ Pas d'éditeur de lien
  - ⊕ Pas de `Makefile` (je l'ai déjà dit ?)
  - ⊕ Les chemins pour la compilation, pour l'édition de liens et pour l'exécution sont les mêmes.
  - ⊖ Performance : des accès disques (et même réseau) cachés au milieu de votre programme

Les trucs reposants

- *Une classe, un fichier*
  - ⊕ Pas de fichiers `.h`
  - ⊕ La doc du code n'est plus le fichier `.h`, c'est une vraie doc crée par `javadoc`
  - ⊕ Pas besoin de `Makefile` ni d'`autoconf`
  - ⊕ Pas de bug dû au préprocesseur
  - ⊖ Si cela se trouve, c'est même mieux que Caml
- *Édition de lien dynamique*
  - Lorsque la machine virtuelle a besoin de créer un objet de classe `Toto`, elle charge `Toto.class`
  - Il faut qu'elle sache où le trouver (on verra plus loin)
  - ⊕ Pas d'éditeur de lien
  - ⊕ Pas de `Makefile` (je l'ai déjà dit ?)
  - ⊕ Les chemins pour la compilation, pour l'édition de liens et pour l'exécution sont les mêmes.
  - ⊖ Performance : des accès disques (et même réseau) cachés au milieu de votre programme

Les trucs reposants

- *Une classe, un fichier*
  - ⊕ Pas de fichiers `.h`
  - ⊕ La doc du code n'est plus le fichier `.h`, c'est une vraie doc crée par `javadoc`
  - ⊕ Pas besoin de `Makefile` ni d'`autoconf`
  - ⊕ Pas de bug dû au préprocesseur
  - ⊖ Si cela se trouve, c'est même mieux que Caml
- *Édition de lien dynamique*
  - Lorsque la machine virtuelle a besoin de créer un objet de classe `Toto`, elle charge `Toto.class`
  - Il faut qu'elle sache où le trouver (on verra plus loin)
  - ⊕ Pas d'éditeur de lien
  - ⊕ Pas de `Makefile` (je l'ai déjà dit ?)
  - ⊕ Les chemins pour la compilation, pour l'édition de liens et pour l'exécution sont les mêmes.
  - ⊖ Performance : des accès disques (et même réseau) cachés au milieu de votre programme

Les trucs reposants

- *Une classe, un fichier*
  - ⊕ Pas de fichiers `.h`
  - ⊕ La doc du code n'est plus le fichier `.h`, c'est une vraie doc crée par `javadoc`
  - ⊕ Pas besoin de `Makefile` ni d'`autoconf`
  - ⊕ Pas de bug dû au préprocesseur
  - ⊖ Si cela se trouve, c'est même mieux que Caml
- *Édition de lien dynamique*
  - Lorsque la machine virtuelle a besoin de créer un objet de classe `Toto`, elle charge `Toto.class`
  - Il faut qu'elle sache où le trouver (on verra plus loin)
  - ⊕ Pas d'éditeur de lien
  - ⊕ Pas de `Makefile` (je l'ai déjà dit ?)
  - ⊕ Les chemins pour la compilation, pour l'édition de liens et pour l'exécution sont les mêmes.
  - ⊖ Performance : des accès disques (et même réseau) cachés au milieu de votre programme

Les trucs reposants

- *Une classe, un fichier*
  - ⊕ Pas de fichiers `.h`
  - ⊕ La doc du code n'est plus le fichier `.h`, c'est une vraie doc crée par `javadoc`
  - ⊕ Pas besoin de `Makefile` ni d'`autoconf`
  - ⊕ Pas de bug dû au préprocesseur
  - ⊖ Si cela se trouve, c'est même mieux que Caml
- *Édition de lien dynamique*
  - Lorsque la machine virtuelle a besoin de créer un objet de classe `Toto`, elle charge `Toto.class`
  - Il faut qu'elle sache où le trouver (on verra plus loin)
  - ⊕ Pas d'éditeur de lien
  - ⊕ Pas de `Makefile` (je l'ai déjà dit ?)
  - ⊕ Les chemins pour la compilation, pour l'édition de liens et pour l'exécution sont les mêmes.
  - ⊖ Performance : des accès disques (et même réseau) cachés au milieu de votre programme

Les trucs reposants

- *Une classe, un fichier*
  - ⊕ Pas de fichiers `.h`
  - ⊕ La doc du code n'est plus le fichier `.h`, c'est une vraie doc crée par `javadoc`
  - ⊕ Pas besoin de `Makefile` ni d'`autoconf`
  - ⊕ Pas de bug dû au préprocesseur
  - ⊖ Si cela se trouve, c'est même mieux que Caml
- *Édition de lien dynamique*
  - Lorsque la machine virtuelle a besoin de créer un objet de classe `Toto`, elle charge `Toto.class`
  - Il faut qu'elle sache où le trouver (on verra plus loin)
  - ⊕ Pas d'éditeur de lien
  - ⊕ Pas de `Makefile` (je l'ai déjà dit ?)
  - ⊕ Les chemins pour la compilation, pour l'édition de liens et pour l'exécution sont les mêmes.
  - ⊖ Performance : des accès disques (et même réseau) cachés au milieu de votre programme

Les trucs reposants

- *Une classe, un fichier*
  - ⊕ Pas de fichiers `.h`
  - ⊕ La doc du code n'est plus le fichier `.h`, c'est une vraie doc crée par `javadoc`
  - ⊕ Pas besoin de `Makefile` ni d'`autoconf`
  - ⊕ Pas de bug dû au préprocesseur
  - ⊖ Si cela se trouve, c'est même mieux que Caml
- *Édition de lien dynamique*
  - Lorsque la machine virtuelle a besoin de créer un objet de classe `Toto`, elle charge `Toto.class`
  - Il faut qu'elle sache où le trouver (on verra plus loin)
  - ⊕ Pas d'éditeur de lien
  - ⊕ Pas de `Makefile` (je l'ai déjà dit ?)
  - ⊕ Les chemins pour la compilation, pour l'édition de liens et pour l'exécution sont les mêmes.
  - ⊖ Performance : des accès disques (et même réseau) cachés au milieu de votre programme

Applet et application

applet (en français *appliquette* ou *programmouillette*) :

- destiné à tourner dans une fenêtre de navigateur chez des gens qu'on ne connait même pas
- dérive d'une classe qui en principe protège la machine hôte *(sandbox)*
  - pas d'écriture ni de lecture du disque local
  - par contre accès en lecture à tous les fichiers du Ternet par leur *url*

application (en français application)

- pas les restriction précédentes
- un navigateur n'a pas plus le droit de la lancer qu'un autre exécutable

Applications et appliquettes partagent le gros de la bibliothèque standard, notamment l'*awt (abstract windowing toolkit)*.

Applet et application

applet (en français *appliquette* ou *programmouillette*) :

- destiné à tourner dans une fenêtre de navigateur chez des gens qu'on ne connait même pas
- dérive d'une classe qui en principe protège la machine hôte *(sandbox)*
  - pas d'écriture ni de lecture du disque local
  - par contre accès en lecture à tous les fichiers du Ternet par leur *url*

application (en français application)

- pas les restriction précédentes
- un navigateur n'a pas plus le droit de la lancer qu'un autre exécutable

Applications et appliquettes partagent le gros de la bibliothèque standard, notamment l'*awt (abstract windowing toolkit)*.

Applet et application

applet (en français *appliquette* ou *programmouillette*) :

- destiné à tourner dans une fenêtre de navigateur chez des gens qu'on ne connait même pas
- dérive d'une classe qui en principe protège la machine hôte *(sandbox)*
  - pas d'écriture ni de lecture du disque local
  - par contre accès en lecture à tous les fichiers du Ternet par leur *url*

application (en français application)

- pas les restriction précédentes
- un navigateur n'a pas plus le droit de la lancer qu'un autre exécutable

Applications et appliquettes partagent le gros de la bibliothèque standard, notamment l'*awt (abstract windowing toolkit)*.

Au fait

JavaScript n'a rien à voir avec Java à part la syntaxe superficielle : c'est un langage interprété, plein de trous de sécurité, pas OO pour un sou, et qu'on laissera volontiers aux authentiques kakous.

Javadoc

- Un outil qui prend du code bien documenté, et construit une doc html toute jolie
- Les commentaires pour Javadoc sont entre `/** ... */`
- Un commentaire de ce type par méthode, attribut, classe, etc.
- Ligne de commande : `javadoc Toto.java`
- Les docs de référence des classes standard du langage sont construites comme cela

Exemple de commentaire de l'an dernier :

```
/**
* classe Cartes
*/
public class Cartes extends Remote ...
```

Javadoc

- Un outil qui prend du code bien documenté, et construit une doc html toute jolie
- Les commentaires pour Javadoc sont entre `/** ... */`
- Un commentaire de ce type par méthode, attribut, classe, etc.
- Ligne de commande : `javadoc Toto.java`
- Les docs de référence des classes standard du langage sont construites comme cela

Exemple de commentaire de l'an dernier :

```
/**
* classe Cartes
*/
public class Cartes extends Remote ...
```

Paquetages

- Un paquetage c'est comme un module en Caml.
- Les paquetages sont organisés hiérarchiquement en répertoires.
  Rien à voir avec la hiérarchie des classes.
- Exemples tirés des bibliothèques standard :

`java.applet`	`java.math`
`java.io`	`java.awt`
`java.awt.image`	`java.awt.event`

- On déclare qu'une classe `Toto` fait partie d'un paquetage `projetLala.tata` en mettant tout au début de `Toto.java` :

```
package projetLala.tata;
```

- En l'absence d'une telle ligne, la classe fait partie du paquetage par défaut, constitué de
  - l'ensemble des classes dans le répertoire courant
  - l'ensemble des classes accessibles par la variable *CLASSPATH*
- On peut ranger un paquetage dans une archive zip, qui s'appellera d'ailleurs jar, mais il faut lire le manuel.

Paquetages

- Un paquetage c'est comme un module en Caml.
- Les paquetages sont organisés hiérarchiquement en répertoires.
  Rien à voir avec la hiérarchie des classes.
- Exemples tirés des bibliothèques standard :

`java.applet`	`java.math`
`java.io`	`java.awt`
`java.awt.image`	`java.awt.event`

- On déclare qu'une classe `Toto` fait partie d'un paquetage `projetLala.tata` en mettant tout au début de `Toto.java` :

```
package projetLala.tata;
```

- En l'absence d'une telle ligne, la classe fait partie du paquetage par défaut, constitué de
  - l'ensemble des classes dans le répertoire courant
  - l'ensemble des classes accessibles par la variable *CLASSPATH*
- On peut ranger un paquetage dans une archive zip, qui s'appellera d'ailleurs jar, mais il faut lire le manuel.

Paquetages

- Un paquetage c'est comme un module en Caml.
- Les paquetages sont organisés hiérarchiquement en répertoires.
  Rien à voir avec la hiérarchie des classes.
- Exemples tirés des bibliothèques standard :

  `java.applet` `java.math`
  `java.io` `java.awt`
  `java.awt.image` `java.awt.event`
- On déclare qu'une classe `Toto` fait partie d'un paquetage `projetLala.tata` en mettant tout au début de `Toto.java` :

```
package projetLala.tata;
```

- En l'absence d'une telle ligne, la classe fait partie du paquetage par défaut, constitué de
  - l'ensemble des classes dans le répertoire courant
  - l'ensemble des classes accessibles par la variable *CLASSPATH*
- On peut ranger un paquetage dans une archive zip, qui s'appellera d'ailleurs jar, mais il faut lire le manuel.

Paquetages

- Un paquetage c'est comme un module en Caml.
- Les paquetages sont organisés hiérarchiquement en répertoires.
  Rien à voir avec la hiérarchie des classes.
- Exemples tirés des bibliothèques standard :

  `java.applet` `java.math`
  `java.io` `java.awt`
  `java.awt.image` `java.awt.event`
- On déclare qu'une classe `Toto` fait partie d'un paquetage `projetLala.tata` en mettant tout au début de `Toto.java` :

```
package projetLala.tata;
```

- En l'absence d'une telle ligne, la classe fait partie du paquetage par défaut, constitué de
  - l'ensemble des classes dans le répertoire courant
  - l'ensemble des classes accessibles par la variable *CLASSPATH*
- On peut ranger un paquetage dans une archive zip, qui s'appellera d'ailleurs jar, mais il faut lire le manuel.

Paquetages

- Un paquetage c'est comme un module en Caml.
- Les paquetages sont organisés hiérarchiquement en répertoires.
  Rien à voir avec la hiérarchie des classes.
- Exemples tirés des bibliothèques standard :

  `java.applet` `java.math`
  `java.io` `java.awt`
  `java.awt.image` `java.awt.event`
- On déclare qu'une classe `Toto` fait partie d'un paquetage `projetLala.tata` en mettant tout au début de `Toto.java` :

```
package projetLala.tata;
```

- En l'absence d'une telle ligne, la classe fait partie du paquetage par défaut, constitué de
  - l'ensemble des classes dans le répertoire courant
  - l'ensemble des classes accessibles par la variable *CLASSPATH*
- On peut ranger un paquetage dans une archive zip, qui s'appellera d'ailleurs jar, mais il faut lire le manuel.

Paquetages

- Un paquetage c'est comme un module en Caml.
- Les paquetages sont organisés hiérarchiquement en répertoires.
  Rien à voir avec la hiérarchie des classes.
- Exemples tirés des bibliothèques standard :

  `java.applet` `java.math`
  `java.io` `java.awt`
  `java.awt.image` `java.awt.event`
- On déclare qu'une classe `Toto` fait partie d'un paquetage `projetLala.tata` en mettant tout au début de `Toto.java` :

```
package projetLala.tata;
```

- En l'absence d'une telle ligne, la classe fait partie du paquetage par défaut, constitué de
  - l'ensemble des classes dans le répertoire courant
  - l'ensemble des classes accessibles par la variable *CLASSPATH*
- On peut ranger un paquetage dans une archive zip, qui s'appellera d'ailleurs jar, mais il faut lire le manuel.

Paquetages

- Un paquetage c'est comme un module en Caml.
- Les paquetages sont organisés hiérarchiquement en répertoires.
  Rien à voir avec la hiérarchie des classes.
- Exemples tirés des bibliothèques standard :

  `java.applet` `java.math`
  `java.io` `java.awt`
  `java.awt.image` `java.awt.event`
- On déclare qu'une classe `Toto` fait partie d'un paquetage `projetLala.tata` en mettant tout au début de `Toto.java` :

```
package projetLala.tata;
```

- En l'absence d'une telle ligne, la classe fait partie du paquetage par défaut, constitué de
  - l'ensemble des classes dans le répertoire courant
  - l'ensemble des classes accessibles par la variable *CLASSPATH*
- On peut ranger un paquetage dans une archive zip, qui s'appellera d'ailleurs jar, mais il faut lire le manuel.

Paquetages et nommages

- Le nom complet d'une classe est `paquetage.Classe`
- Unicité planétaire des noms par l'URL renversée : `com.projetMIM2002.www.lala.test`
- Le nom complet d'un membre est `paquetage.Classe.membre`
- Si on a la flemme de taper tout cela tout le temps dans `Toto.java`, on peut *importer* une fois pour toutes, au début de ce fichier,
  - une classe : `import java.lang.Math;`
  - ou bien toutes les classes d'un paquetage : `import java.lang.*;`
- On n'est pas dispensé de taper `Classe.membre` ou `objet.membre`... Exemple `Math.sin(x)`.

Paquetages et nommages

- Le nom complet d'une classe est `paquetage.Classe`
- Unicité planétaire des noms par l'URL renversée : `com.projetMIM2002.www.lala.test`
- Le nom complet d'un membre est `paquetage.Classe.membre`
  - la fonction sinus : `java.lang.Math.sin(x)`
  - `com.projetMIM2002.www.lala.test.membre` pour un membre de `test`
- Si on a la flemme de taper tout cela tout le temps dans `Toto.java`, on peut *importer* une fois pour toutes, au début de ce fichier,
  - une classe : `import java.lang.Math;`
  - ou bien toutes les classes d'un paquetage : `import java.lang.*;`
- On n'est pas dispensé de taper `Classe.membre` ou `objet.membre`... Exemple `Math.sin(x)`.

Paquetages et nommages

- Le nom complet d'une classe est `paquetage.Classe`
- Unicité planétaire des noms par l'URL renversée : `com.projetMIM2002.www.lala.test`
- Le nom complet d'un membre est `paquetage.Classe.membre`
  - la fonction sinus : `java.lang.Math.sin(x)`
  - la variable à l'unicité planétaire : `com.projetMIM2002.www.lala.test.RandomTest.nombreErreurs`
- Si on a la flemme de taper tout cela tout le temps dans `Toto.java`, on peut *importer* une fois pour toutes, au début de ce fichier,
  - une classe : `import java.lang.Math;`
  - ou bien toutes les classes d'un paquetage : `import java.lang.*;`
- On n'est pas dispensé de taper `Classe.membre` ou `objet.membre`... Exemple `Math.sin(x)`.

Paquetages et nommages

- Le nom complet d'une classe est `paquetage.Classe`
- Unicité planétaire des noms par l'URL renversée : `com.projetMIM2002.www.lala.test`
- Le nom complet d'un membre est `paquetage.Classe.membre`
  - la fonction sinus : `java.lang.Math.sin(x)`
  - la variable à l'unicité planétaire : `com.projetMIM2002.www.lala.test.RandomTest.nombreErreurs`
- Si on a la flemme de taper tout cela tout le temps dans `Toto.java`, on peut *importer* une fois pour toutes, au début de ce fichier,
  - une classe : `import java.lang.Math;`
  - ou bien toutes les classes d'un paquetage : `import java.lang.*;`
- On n'est pas dispensé de taper `Classe.membre` ou `objet.membre`... Exemple `Math.sin(x)`.

Paquetages et nommages

- Le nom complet d'une classe est `paquetage.Classe`
- Unicité planétaire des noms par l'URL renversée : `com.projetMIM2002.www.lala.test`
- Le nom complet d'un membre est `paquetage.Classe.membre`
  - la fonction sinus : `java.lang.Math.sin(x)`
  - la variable à l'unicité planétaire : `com.projetMIM2002.www.lala.test.RandomTest.nombreErreurs`
- Si on a la flemme de taper tout cela tout le temps dans `Toto.java`, on peut *importer* une fois pour toutes, au début de ce fichier,
  - une classe : `import java.lang.Math;`
  - ou bien toutes les classes d'un paquetage : `import java.lang.*;`
- On n'est pas dispensé de taper `Classe.membre` ou `objet.membre`... Exemple `Math.sin(x)`.

Paquetages et nommages

- Le nom complet d'une classe est `paquetage.Classe`
- Unicité planétaire des noms par l'URL renversée : `com.projetMIM2002.www.lala.test`
- Le nom complet d'un membre est `paquetage.Classe.membre`
  - la fonction sinus : `java.lang.Math.sin(x)`
  - la variable à l'unicité planétaire : `com.projetMIM2002.www.lala.test.RandomTest.nombreErreurs`
- Si on a la flemme de taper tout cela tout le temps dans `Toto.java`, on peut *importer* une fois pour toutes, au début de ce fichier,
  - une classe : `import java.lang.Math;`
  - ou bien toutes les classes d'un paquetage : `import java.lang.*;`
- On n'est pas dispensé de taper `Classe.membre` ou `objet.membre`... Exemple `Math.sin(x)`.

Paquetages et nommages

- Le nom complet d'une classe est `paquetage.Classe`
- Unicité planétaire des noms par l'URL renversée : `com.projetMIM2002.www.lala.test`
- Le nom complet d'un membre est `paquetage.Classe.membre`
  - la fonction sinus : `java.lang.Math.sin(x)`
  - la variable à l'unicité planétaire : `com.projetMIM2002.www.lala.test.RandomTest.nombreErreurs`
- Si on a la flemme de taper tout cela tout le temps dans `Toto.java`, on peut *importer* une fois pour toutes, au début de ce fichier,
  - une classe : `import java.lang.Math;`
  - ou bien toutes les classes d'un paquetage : `import java.lang.*;`
- On n'est pas dispensé de taper `Classe.membre` ou `objet.membre`... Exemple `Math.sin(x)`.

Quelques conventions

- Tout le monde appelle ses classes avec une majuscule. Du coup les fichiers aussi.
- Les membres de la plupart des programmeurs sont minuscules.
- Vous faites bien ce que vous voudrez.

Portée des classes, portée des identificateurs

- Dans le corps d'une méthode, c'est comme en C.
- Dans le corps d'une classe, un membre peut être
  - `public` : tout le monde le voit même en dehors de la classe
  - `private` : visible uniquement à l'intérieur de la classe
  - `protected` : visible dans la classe, ses sous-classes, et le paquetage
  - rien du tout : visible dans le paquetage
- Dans un paquetage, une classe peut-être
  - `public` : tout le monde le voit même en dehors du paquetage
  - rien du tout : visible dans le paquetage seulement
- Vous trouverez plein d'exemples sur le Ternet.

Portée des classes, portée des identificateurs

- Dans le corps d'une méthode, c'est comme en C.

- Dans le corps d'une classe, un membre peut être
  - `public` : tout le monde le voit même en dehors de la classe
  - `private` : visible uniquement à l'intérieur de la classe
  - `protected` : visible dans la classe, ses sous-classes, et le paquetage
  - rien du tout : visible dans le paquetage

- Dans un paquetage, une classe peut-être
  - `public` : tout le monde le voit même en dehors du paquetage
  - rien du tout : visible dans le paquetage seulement

- Vous trouverez plein d'exemples sur le Ternet.

Portée des classes, portée des identificateurs

- Dans le corps d'une méthode, c'est comme en C.
- Dans le corps d'une classe, un membre peut être
  - `public` : tout le monde le voit même en dehors de la classe
  - `private` : visible uniquement à l'intérieur de la classe
  - `protected` : visible dans la classe, ses sous-classes, et le paquetage
  - rien du tout : visible dans le paquetage
- Dans un paquetage, une classe peut-être
  - `public` : tout le monde le voit même en dehors du paquetage
  - rien du tout : visible dans le paquetage seulement
- Vous trouverez plein d'exemples sur le Ternet.

Portée des classes, portée des identificateurs

- Dans le corps d'une méthode, c'est comme en C.
- Dans le corps d'une classe, un membre peut être
  - `public` : tout le monde le voit même en dehors de la classe
  - `private` : visible uniquement à l'intérieur de la classe
  - `protected` : visible dans la classe, ses sous-classes, et le paquetage
  - rien du tout : visible dans le paquetage
- Dans un paquetage, une classe peut-être
  - `public` : tout le monde le voit même en dehors du paquetage
  - rien du tout : visible dans le paquetage seulement
- Vous trouverez plein d'exemples sur le Ternet.

Portée des classes, portée des identificateurs

- Dans le corps d'une méthode, c'est comme en C.
- Dans le corps d'une classe, un membre peut être
  - `public` : tout le monde le voit même en dehors de la classe
  - `private` : visible uniquement à l'intérieur de la classe
  - `protected` : visible dans la classe, ses sous-classes, et le paquetage
  - rien du tout : visible dans le paquetage
- Dans un paquetage, une classe peut-être
  - `public` : tout le monde le voit même en dehors du paquetage
  - rien du tout : visible dans le paquetage seulement
- Vous trouverez plein d'exemples sur le Ternet.

Les faciles

- Entiers *signés* de différentes tailles : `byte` (8 bits), `short` (16), `int` (32), `long` (64)
- Nombres en virgule flottante : `float` et `double`
- `boolean` qui vaut `true` ou `false`
- `char` est un caractère Unicode (sur 16 bits !)

- Tous ces types ont une valeur initiale spécifiée par le langage, mais `javac` fait des warning si on ne les initialise pas.

```
class HelloHello {
 public static void main(String[] args) {
  int i;
  for(i=0; i<10; i++) {
   System.out.println("Hello, java, " + i + " fois?");
  }
 }
}
```

Les objets

```
import java.awt.*;

class HelloLaFenetre {
 public static void main(String[] args) {
  Frame objet_fenetre;

  objet_fenetre = new Frame("Hello, java ?") ;
  objet_fenetre.setSize(300,100);
  objet_fenetre.setVisible(true);
 }
}
```

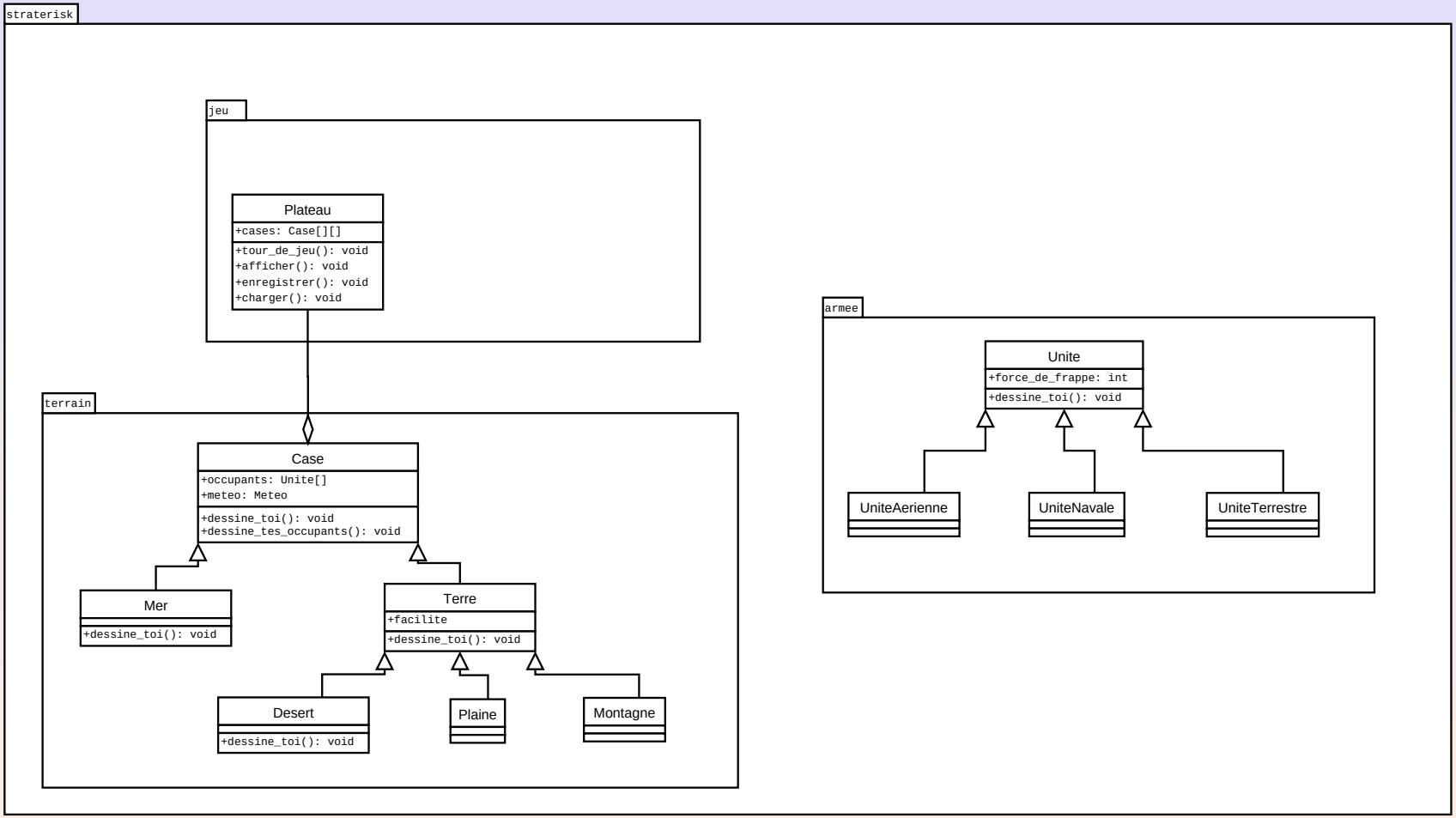
La vraie classe

```
import java.awt.*;

public class ObjetHello {

  private Frame fenetre;

  public void construit(String titre){
    fenetre = new Frame(titre) ;
    fenetre.setSize(300,100);
    fenetre.setVisible(true);
    }

  //constructeurs
    public ObjetHello(){
      construit("Hello, java ?");
    }

    public ObjetHello(String titre){
      construit(titre);
    }
    // Pas besoin de main() ici
}
```

```
//import java.awt.*; // plus besoin ici

public class ProjetHello {

  public static void main(String[] args) {
    // un objet que c'est moi qui l'ai fait
    ObjetHello uoqcmqlaf;

    uoqcmqlaf = new ObjetHello();

    uoqcmqlaf = new ObjetHello("Youpi!");


  }
}
```

Héritage

Reprenons cet UML bâclé :



Héritage

et passons-le à dia2code

```
package straterisk.terrain;

import straterisk.jeu.Plateau;

public abstract class Case {
  /** Attributes */
  public Unite[] occupants;
  public Meteo meteo;
  /** Associations */
  private Plateau;
  /**
   * Operation
   *
   */
  abstract public void dessine_toi (  );
  /**
   * Operation
   *
   */
  public void dessine_tes_occupants (  ){
  }
}
```

```
package straterisk.terrain;

import straterisk.terrain.Case;

public class Terre extends Case {
  /** Attributes */
  public  facilite;
  /**
   * Operation
   *
   */
  public void dessine_toi (  ){
  }
}
```

```
package straterisk.terrain;

import straterisk.terrain.Terre;

public class Desert extends Terre {
  /**
   * Operation
   *
   */
  public void dessine_toi (  ){
  }
}
```

Remarques sordides

`dia2code` c'est pas encore cela

- Il gère bien les `import` et `package` dans le code source, mais ne construit pas la hiérarchie des répertoires
- La version du CRI est un peu vieille
- Utilisable uniquement en phase initiale

Mais bon, cela vous oblige à bien réfléchir votre modèle objet avant de coder...

Retour aux types de base : les bizarres

- `String` est une classe d'objets (majuscule...) mais avec du sucre syntaxique dans le langage pour le constructeur, la concaténation (+), ...

```
int i,j;
String errorMessage;
(...)
errorMessage="Feature not implemented, because we
started the project two days before the deadline";
```

  Méthodes d'une chaîne : `errorMessage.length()`, et plein d'autres

- Il y a aussi des chaînes de taille variable (`StringBuffer`), etc.

Retour aux types : les bizarres (2)

Les tableaux sont aussi des objets avec du sucre syntaxique et sémantique (classe paramétrée).

```
public class SpaceHello {

  public static void main(String[] args) {
  ObjetHello[] plein_de_fenetres;

  if(args.length == 0)
    System.err.println("Usage: java SpaceHello text ");
  else
    {
      int i;
      plein_de_fenetres = new ObjetHello[args.length];
      for(i=0; i<args.length; i++)
         plein_de_fenetres[i] = new ObjetHello(args[i]);
    }
  }
}
```

Références

- Le site de Sun `http ://java.sun.com`
  - Des tutoriels
  - La doc de référence sur le langage (on s'en passe bien)
  - La doc de références sur les classes standard (à parcourir absolument !)
  - Le tout téléchargeable pour pouvoir partir en vacances avec
- D'autres sites, par exemple `http ://java.developpez.com`
- Plein de bouquins à la bibliothèque
- Celui de Brondeau en français
  - ⊕ court
  - ⊕ en français
  - ⊖ un peu lège sur l'OO

Semaine prochaine

- J'ai la liste de tous les groupes
- J'ai un sujet détaillé par groupe
- J'ai un (échéancier + cahier des charges) par groupe
- Présentation de Netbeans
- On parlera de Design Patterns ?
- On parlera de linkage statique Vs. dynamique ?