Université Nice SOPHIA ANTIPOLIS

Génie Logiciel Orienté Objets

Philippe Collet

Master 1 Informatique

2005-2006

http://deptinfo.unice.fr/twiki/bin/view/Minfo/GLOO

Génie logiciel : organisation générale

Génie Logiciel et supports de Programmation *Design Patterns,* Réflexivité, Tests OO…

M1

Spécifications Spec. OO

Analyse et Conception Orientée objets

Environnement de Programmation (outils pour le Génie Logiciel)

L3

GLOO : Organisation

- Introduction, approches de devt, outil ant
- Introspection, réflexivité
- V&V objet, JUnit
- Assertion, JML & JUnit
- Généricité
- Héritage
- Design patterns
- Composants et framework (Java Beans)
- Chargement dynamique de classes
- Analyse de performances, conclusion

- Évaluation
  - Contrôle continu (30 %) : TP long
  - Examen (70 %)

GLOO : objectifs

- Maîtriser les techniques de spécification et de test pour le génie logiciel, en se focalisant sur l'approche par objets.
- Utiliser les techniques orientées objets et composants pour le génie logiciel : héritage vs. composition, introduction aux patrons de conception, réflexivité, chargement dynamique de classes.

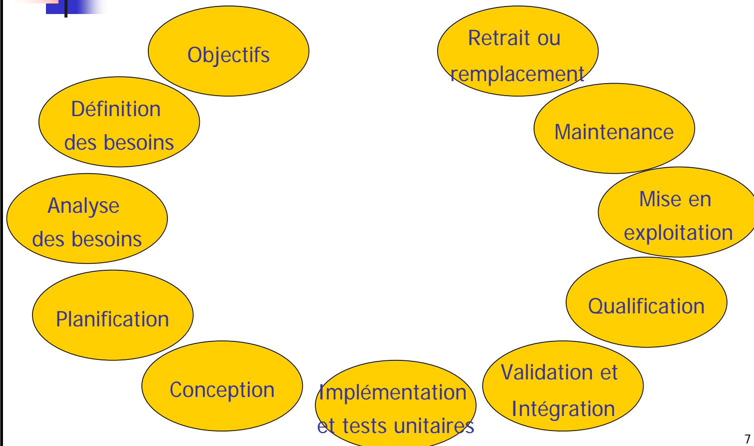
Cycle de vie du logiciel

- Les phases du cycle de vie
- Les modèles de développement

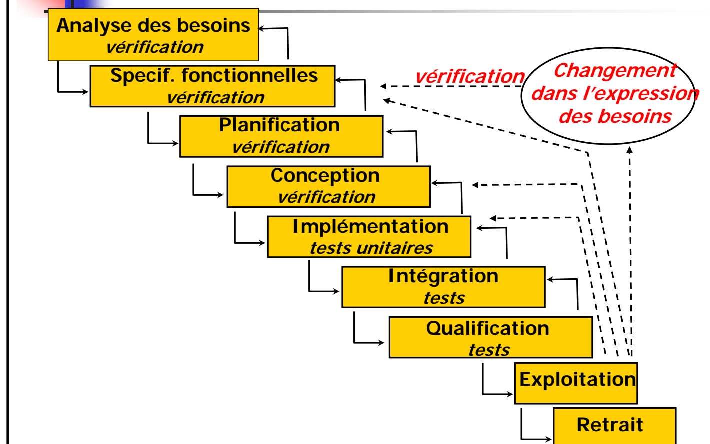
Notion de cycle de vie

- Description d'un processus pour :
  - la création d 'un produit
  - sa distribution sur un marché
  - son retrait
- Cycle de vie et assurance qualité
  - Validation : le bon produit ?
  - Vérification : le produit correct ?

Les phases du cycle de vie



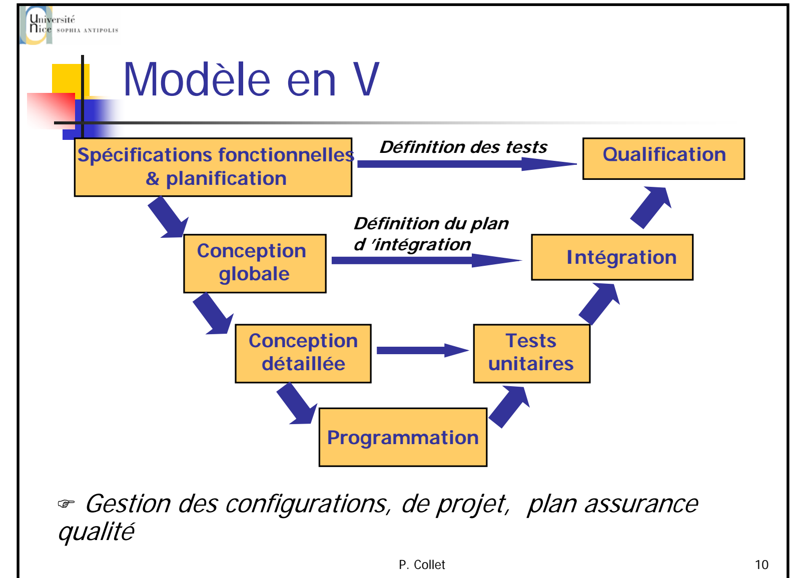
Modèle en cascade (1970)



Problèmes du modèle en cascade

- Les vrais projets suivent rarement un développement séquentiel
- Établir tous les besoins au début d'un projet est difficile
- Le produit apparaît tard
- Seulement applicable pour les projets qui sont bien compris et maîtrisés

Modèle en V

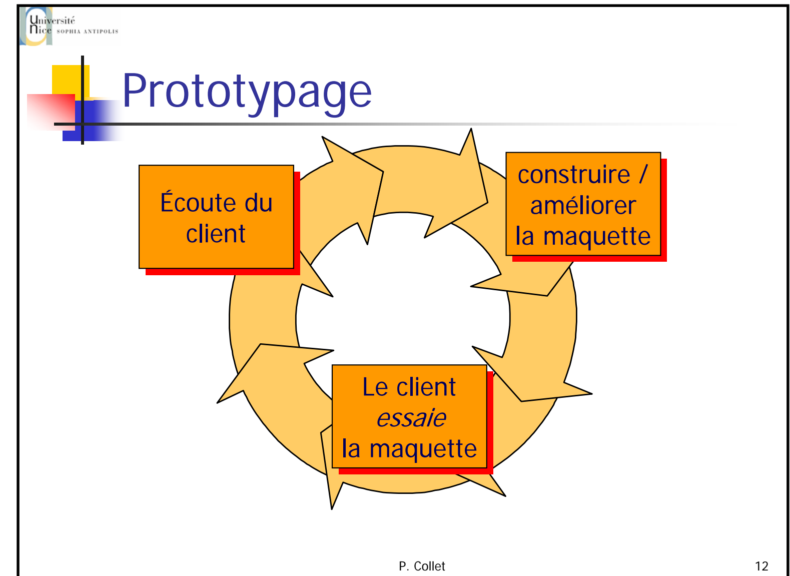


☞ *Gestion des configurations, de projet, plan assurance qualité*

Comparaison

- Le cycle en V
  - permet une meilleure anticipation
  - évite les retours en arrière
- Mais
  - le cadre de développement est rigide
  - la durée est souvent trop longue
  - le produit apparaît très tard

Prototypage



Prototypage, RAD

RAD : *Rapid Application Development*

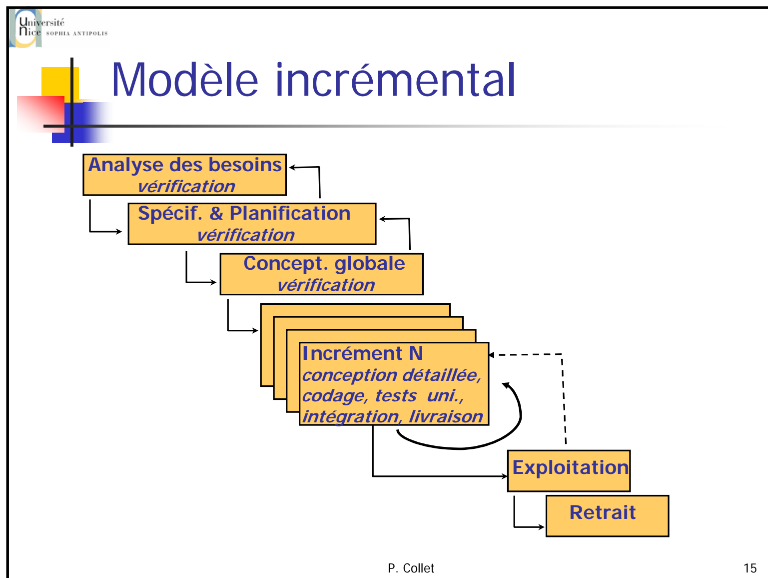
- Discuter et interagir avec l'utilisateur
- Vérifier l 'efficacité réelle d 'un algorithme
- Vérifier des choix spécifiques d 'IHM
- Souvent utilisé pour identifier les besoins
  - Prototype jetable (moins de risque ?)
- Souvent implémenté par des générateurs de code
  - Prototype évolutif

Prototypage, RAD (suite)

- Mais :
  - Les objectifs sont uniquement généraux
  - Prototyper n'est pas spécifier
  - Les décisions rapides sont rarement de bonnes décisions
  - Le prototype évolutif donne-t-il le produit demandé ?
  - Les générateurs de code produisent-ils du code assez efficace ?

☞ *Projets petits ou à courte durée de vie*

Modèle incrémental



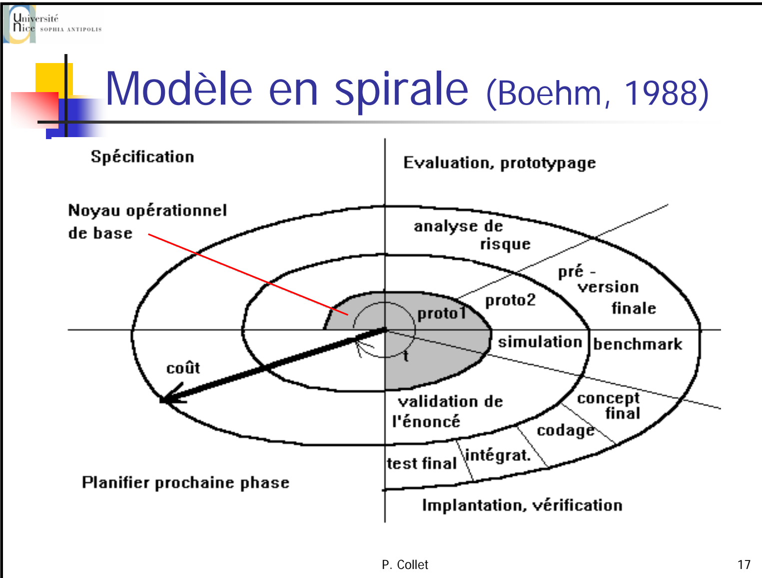
Le développement incrémental

- combine des éléments des modèles linéaires et du prototypage
- produit des incréments *livrables*
- se concentre sur un produit opérationnel (pas de prototype jetable)
- peut être utilisé quand il n'y a pas assez de ressources disponibles pour une livraison à temps

☞ *Le premier incrément est souvent le noyau*

☞ *Les incréments aident à gérer les risques techniques (matériel non disponible)*

Modèle en spirale (Boehm, 1988)



Modèle en spirale (suite)

- Spécification : communiquer avec le client
- Analyse de risque : évaluation des risques techniques et des risques de gestion
- Implémentation et vérification : construire, tester, installer et fournir un support utilisateur
- Validation: obtenir des *retours*
- Planification : définir les ressources, la répartition dans le temps

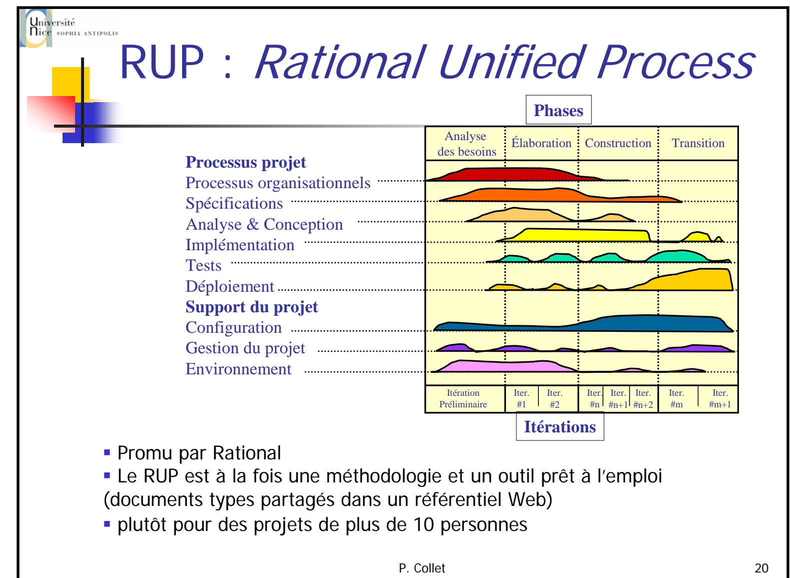
Modèle en spirale (suite)

- Couplage de la nature itérative du prototypage avec les aspects systématiques et contrôlés du modèle en cascade
- Les premières itérations peuvent être des modèles *sur papier* ou des prototypes
- Utilisation possible tout au long de la vie du produit

☞ *Réduit les risques si bien appliqué*

☞ *Les augmentent considérablement si le contrôle faiblit*

RUP : *Rational Unified Process*



- Promu par Rational
- Le RUP est à la fois une méthodologie et un outil prêt à l'emploi (documents types partagés dans un référentiel Web)
- plutôt pour des projets de plus de 10 personnes

2TUP : *Two Track Unified Process*

- S'articule autour de l'architecture
- Propose un cycle de développement en Y
- Détaillé dans « UML en action »
- pour des projets de toutes tailles

eXtreme Programming (XP…)

- Ensemble de « Bests Practices » de développement (travail en équipes, transfert de compétences…)
- plutôt pour des projets de moins de 10 personnes

4 Valeurs

- **Communication**
- **Simplicité**
- **Feedback**
- **Courage**

Comparaison des 3 processus dans le vent

	Points forts	Points faibles
RUP	▪ Itératif ▪ Spécifie le dialogue entre les différents intervenants du projet : les livrables, les plannings, les prototypes… ▪ Propose des modèles de documents, et des canevas pour des projets types	▪ Coûteux à personnaliser ▪ Très axé processus, au détriment du développement : peu de place pour le code et la technologie
XP	▪ Itératif ▪ Simple à mettre en œuvre ▪ Fait une large place aux aspects techniques : prototypes, règles de développement, tests… ▪ Innovant: programmation en duo…	▪ Ne couvre pas les phases en amont et en aval au développement : capture des besoins, support, maintenance, tests d'intégration… ▪ Élude la phase d'analyse, si bien qu'on peut dépenser son énergie à faire et défaire ▪ Assez flou dans sa mise en œuvre: quels intervenants, quels livrables ?
2TUP	▪ Itératif ▪ Fait une large place à la technologie et à la gestion du risque ▪ Définit les profils des intervenants, les livrables, les plannings, les prototypes	▪ Plutôt superficiel sur les phases situées en amont et en aval du développement : capture des besoins, support, maintenance, gestion du changement… ▪ Ne propose pas de documents types

Les différents types de projet

Durée	Personnes	Budget	Approche
< à 1 an	1	< 100 K€	Documentation a posteriori Validation par le développeur Vie limitée
Env. 1 an	1 à 5	< 300 à 500 K€	Plusieurs phases (dont conception) Planning, réunions d'avancement Contrôle qualité interne et gestion de versions Prototypage
1 à 2 ans	6 à 15	< 5 M€	Etudes préliminaires et cycle en spirale Documents de suivi et d'anomalie, inspections Gestion de configurations Plans de validation et d'intégration
2 ans et plus	16 et plus	> 5 M€	Procédures de communication Recettes intermédiaires Contrôle qualité permanent Gestion des sous-projets et de la sous-traitance Tests de non-régression Effort de synthèse et base historique

Université Nice SOPHIA ANTIPOLIS

Automatisation de construction en Java : ANT

- D'après le cours sur Ant de Richard Grin (2002)
- http://ant.apache.org/

Introduction

- Syntaxe et options très fournies
- Dans ce cours
  - Version 1.5
  - Pas de syntaxe complète des tâches
  - Ant est toujours distribué avec son manuel
- Projet Open source (fondation Apache)
- La référence pour la construction automatique et **portable** d'applications Java
- Ecrit lui-même en Java

Principes

- Modèle de la commande make
  - un projet
  - des cibles (compile, jar, javadoc,...)
  - La description des cibles et les dépendances entre les cibles sont décrites dans un fichier
  - *Fichier XML, nommé par défaut build.xml*
- Extensible : on peut ajouter ses propres commandes/tâches

build.xml : exemple

```
<project name="hello" default="compile">

  <target name="prepare">
    <mkdir dir="./classes" />
  </target>

  <target name="compile" depends="prepare">
    <javac srcdir="./src"
           destdir="./classes" />
  </target>
</project>
```

Script de construction : structure

- une en-tête XML (avec l'indication optionnelle d'une DTD)
- une entrée **project** qui contient
  - optionnellement, des entrées **property**
  - optionnellement, des entrées path ou classpath
  - une ou plusieurs entrées **target**
  - optionnellement, une entrée **description**
    - Description informelle du projet
    - ```
      <description>
      Ce projet permet de . . .
      . . .
      </description>
      ```


Entrée **project**

- Chaque fichier de construction contient une et une seule entrée project
- Cette entrée peut avoir 3 attributs
  - **name** le nom du projet
  - **default** la cible par défaut (requis)
  - **basedir** le répertoire de base pour les chemins relatifs
    - peut être écrasé par la propriété **basedir**
    - par défaut le répertoire où se trouve le fichier de construction

Les cibles

- Une cible (**target**)
  - correspond à une action qui est décrite dans le fichier
  - peut dépendre d'autres cibles (attribut **depends**)
- Chaque type de cible peut avoir ses propres attributs
- Les attributs communs à toutes les cibles :
  - **name** : le nom de la cible (obligatoire)
  - **description** : si elle apparaît, permet de lister une description de la cible avec l'option **-projecthelp** de l'appel de **ant**
  - **depends** : permet d'indiquer les autres cibles dont dépend une cible

Dépendances de cibles

- On peut indiquer plusieurs cibles dont une cible dépend (**depends A,B,C** par exemple)
  - les cibles seront exécutées dans l'ordre du **depends** (de gauche à droite)
- Dans la gestion des dépendances, les tâches ne sont exécutées qu'une fois :

```
<target name="A"/>
<target name="B" depends="A"/>
<target name="C" depends="A,B"/>
```

A ne sera exécuté qu'une seule fois
```

Comportement sur erreur

- Le plus souvent, une erreur dans une tâche arrête la construction de la cible correspondante
  - Une classe ne compile pas, la cible qui construit le jar s'arrête
- Certaines tâches ne provoque pas d'arrêt
  - On peut leur ajouter un attribut « **`failonerror`** » à **`true`** pour forcer l'arrêt
  - Exemple : la tâche « **`java`** »

Cible d'initialisation

- Il est recommandé d'avoir une cible d'initialisation nommé **`init`** qui contient au moins la tâche **`tstamp`** :
  - ```
    <target name="init">
      <tstamp/>
    </target>
    ```
  - tstamp récupère le temps système et initialise les propriétés **`DSTAMP`** (aaaammjj), **`TSTAMP`** (hhmm), et **`TODAY`** (mois jour année)
- Toutes les cibles liées à la construction de l'application devront dépendre de la cible **`init`** (directement ou non)

Tâches

- Une tâche est une unité d'exécution « élémentaire » pour réaliser une cible
- Attributs possibles :
  - id donne un identificateur unique à la tâche ; cet identificateur peut être utilisé dans le reste du fichier pour désigner la tâche
  - taskname donne un autre nom à la tâche ; ce nom sera utilisé dans les rapports d'exécution
  - description décrit la tâche (texte non formaté)
- Les tâches optionnelles
  - nécessitent une bibliothèque supplémentaire pour être exécutées (fichier .jar à installer)

Tâches (java)

- Ant fournit des tags XML pour les tâches les plus communes en Java :
  - javac, java, rmic, javadoc, jar, unjar, war, unwar, ear

```
<javadoc
  packagenames=« com.bigmoney.pack.*"
  sourcepath="${src}"
  destdir="${doc}/api"
  use="true"/>
```
```

La tâche **javac**

- Compilateur utilisé : propriété **`build.compiler`**
  - par défaut, JDK qui exécute Ant
- Compiler récursivement tous les fichiers java du répertoire des sources
  - Utilisation des dates de dernière modification pour savoir si une classe a besoin d'être recompilée
- Très grand nombre d'attributs : **`srcdir`** (requis), **`classpath`**, **`debug`**, **`optimize`**, **`source`**, **`fork`**…

```
<javac srcdir="${src}" destdir="${build}"
  classpath="xyz.jar" debug="on" />
```


La tâche **java**

- Lance l'exécution d'un programme java
- Attributs :
  - **`classname`** ou **`jar`** pour indiquer la classe à exécuter
  - **`classpath`**, **`fork`**, **`failonerror`**, **`output`**, **`append`**…

```
<java jar="dist/test.jar" fork="true"
        failonerror="true" maxmemory="128m">
    <arg value="-h"/>
    <classpath>
      <pathelement location="dist/test.jar"/>
      <pathelement path="${java.class.path}"/>
    </classpath>
</java>
```


D'autres tâches

- Système :
  - mkdir, delete, copy, move, chmod, touch, get, zip, unzip, tar, untar, gzip, gunzip
- Propriétés :
  - property donne la valeur d'une propriété

```
<property name="jaxp.jar"
      value="./lib/jaxp11/jaxp.jar"/>
```

  - available initialise une propriété si une ressource est disponible (fichier, répertoire, ressource de la JVM)

```
<available classname="fr.unice.Classe"
      property="Class.present"/>
```


D'autres tâches

- Programmation :
  - fail stoppe le processus de construction
  - ant exécute un autre fichier ant (utile s'il y a des sous-projets)
  - antcall appelle une autre cible du fichier de configuration
  - apply, exec exécute des shellscripts et des programmes externes
  - echo affiche un message sur System.out
  - mail envoie un courrier électronique
  - sql exécute une requête SQL en utilisant une source JDBC
  - ftp établit un client FTP pour transmettre des fichiers
  - junit ajoute des tâches liées à l'outil de tests JUnit (optionnel)
  - cvs exécute une commande CVS (optionnel)

Exécution de Ant

- « **ant** » lance Ant en utilisant
  - le fichier **build.xml** du répertoire courant
  - la cible par défaut
    - on peut donner une autre cible en argument
- Options
  - **-buildfile** pour utiliser un autre fichier que **build.xml**
  - **-D*propriété*=*valeur*** pour donner la valeur d'une propriété
  - **-help** affiche les options disponibles
  - **-projecthelp** affiche une description du projet, avec toutes les cibles (*targets*) qui ont une description

Principaux types de données

- **property** : pour paramétrer la construction
- **filelist** : liste de fichiers, sans jokers
- **dirset** : idem **fileset** pour des répertoires
- **fileset** : permet plus de possibilités que **filelist**, en particulier les **patternset**
- **patternset** : utilisent des jokers ; inclus dans **fileset** ou **dirset**
- **filterset** : pour remplacer des token par des valeurs
- **path**, **classpath** : pour donner des chemins tels que PATH ou CLASSPATH

Propriétés

- Chaque projet peut avoir un ensemble de propriétés qui sont utilisées comme des variables dans les attributs des tâches
- ***${prop}*** représente la valeur de la propriété ***prop***
- Il peut y avoir des propriétés locales ou globales (en dehors de toute cible)
- Le nom d'une propriété est de la forme **project.name** ou **build.dir**, sur le modèle des noms de propriétés Java
- Il est sensible à la casse des lettres
- 3 façons de valuer une propriété :
  - tâche **property**
  - tâche **available**
  - au lancement de Ant avec l'option –D : **ant -D***propriété*=*valeur*…

Tâche property

- Plusieurs façons de donner la valeur d'une ou plusieurs propriétés
- Pour une seule propriété :
  - **name** et **value**
  - **name** et **refid**
  - **name** et **location**
- Pour plusieurs propriétés :
  - **file** (donne le nom d'un fichier au format des propriétés Java)
  - **resource** (idem **file** mais recherche dans le *classpath*)

Exemples

```
<available classname="fr.unice.Classe"
    property="Class.present"/>
```

```
<property name="jaxp.jar"
    value="./lib/jaxp11/jaxp.jar"/>
<available file="${jaxp.jar}"
    property="jaxp.jar.present"/>
```

```
<available file="/usr/local/lib" type="dir"
    property="local.lib.present"/>
```

```
<property file="build.properties"/>
```


Propriétés de base

- On peut utiliser toutes les propriétés système Java données par **System.getProperties()** et aussi des propriétés internes à **ant** :
  - **basedir** : le chemin absolu de la racine du projet (mis par l'attribut « **basedir** » du tag « **project** »
  - **ant.file** : le chemin absolu du fichier de construction
  - **ant.version** : version de Ant
  - **ant.java.version** : la version de la JVM

path et classpath

- Des entrées spéciales **path** et **classpath** sont réservées aux noms ou listes de noms de fichiers
- Ces 2 entrées ont la même syntaxe
- Elles peuvent être
  - incluses dans une définition de cible
  - ou au même niveau que les propriétés globales
    - on leur donne un identificateur et on peut les utiliser dans plusieurs cibles

Classpath (ou path)

- Permet d'indiquer le *classpath* :

```
<classpath>
  <pathelement path="${classpath}"/>
  <pathelement location="lib/helper.jar"/>
</classpath>
```

peut contenir plusieurs entrées

ne peut contenir qu'une entrée

- Les éléments sont indiqués par des entrées **pathelement** ou **fileset**