

# *Composants*

---

*JavaBeans*

# *Composants*

---

- ◆ Réutilisation en Génie logiciel
  - Information, connaissance et expérience acquises lors de projets précédents
  - Architectures logicielles
  - Méthodes de développement
  - Programmes, Codes, Algorithmes
    - Classes, Librairies
    - Composants

# *Objets vs Composants*

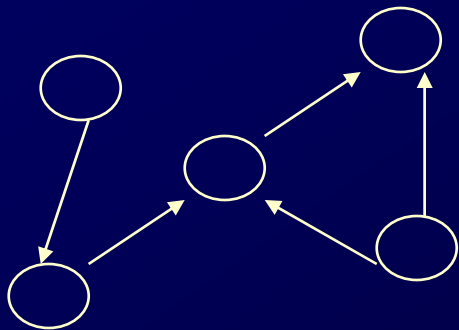
---

- ◆ Objets: construction à petite échelle
  - Encapsulation d'état et définition du comportement
  - Développement Bottom-up
  - Appels de méthodes
  - Interfaces riches
  - Architecture non explicite
- ◆ Composants: construction à grande échelle
  - Encapsulation d'éléments standards
  - Développement Top-down
  - Composition Simple
  - Code Colle (automatiquement généré)
  - Analogie: circuit intégrés

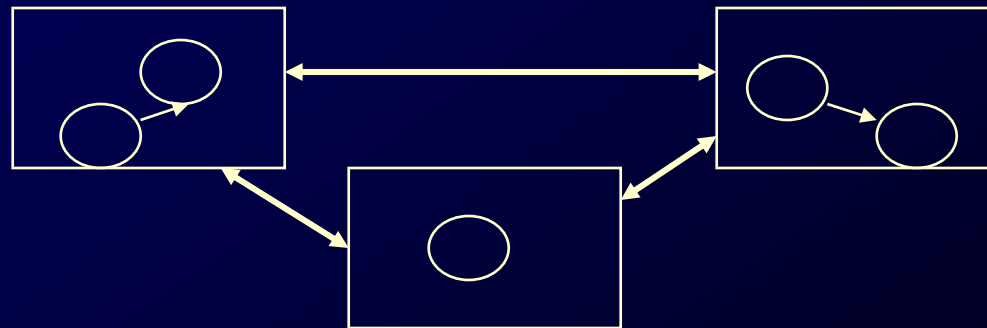
# Objets vs Composants

---

Objets



Composants



# *Composants: Définition*

---

- ◆ **Unité de composition**
  - entité de base pour la construction d'une application
  - définit une abstraction / peut être manipulé dans des outils
- ◆ **Interface contractuelle**
  - signature, pre- post-conditions, contrats
  - communications inter-composants standardisées
- ◆ **Dépendance explicite du contexte**
  - pas de variable globale
- ◆ **Développement incrémental**
  - un composant peut être encapsulé dans un autre composant
- ◆ **Composition**
  - réutilisabilité

# *Composants: Abstraction*

---

- ◆ Unité de construction standard utilisée pour développer des applications
- ◆ Propriétés:
  - Indépendance vis-à-vis de l'application (faible ou forte)
  - Utilisé de l'extérieur
    - On ne sait pas comment il fonctionne à l'intérieur
  - Abstraction générale (en vue de la réutilisabilité)
    - Représente une simplification conceptuelle de ce qui est implémenté
    - Définir suffisamment (mais pas trop) d'opérations

# *Composants: Abstraction*

---

- Qualité
  - Testé, efficace, bien documenté
- Invitation à la réutilisabilité
  - Bonne interface
  - Facile à retrouver
  - Bien documenté
- Flexible
  - Facile à changer en vue d'une adaptation

# *Composants vs Frameworks*

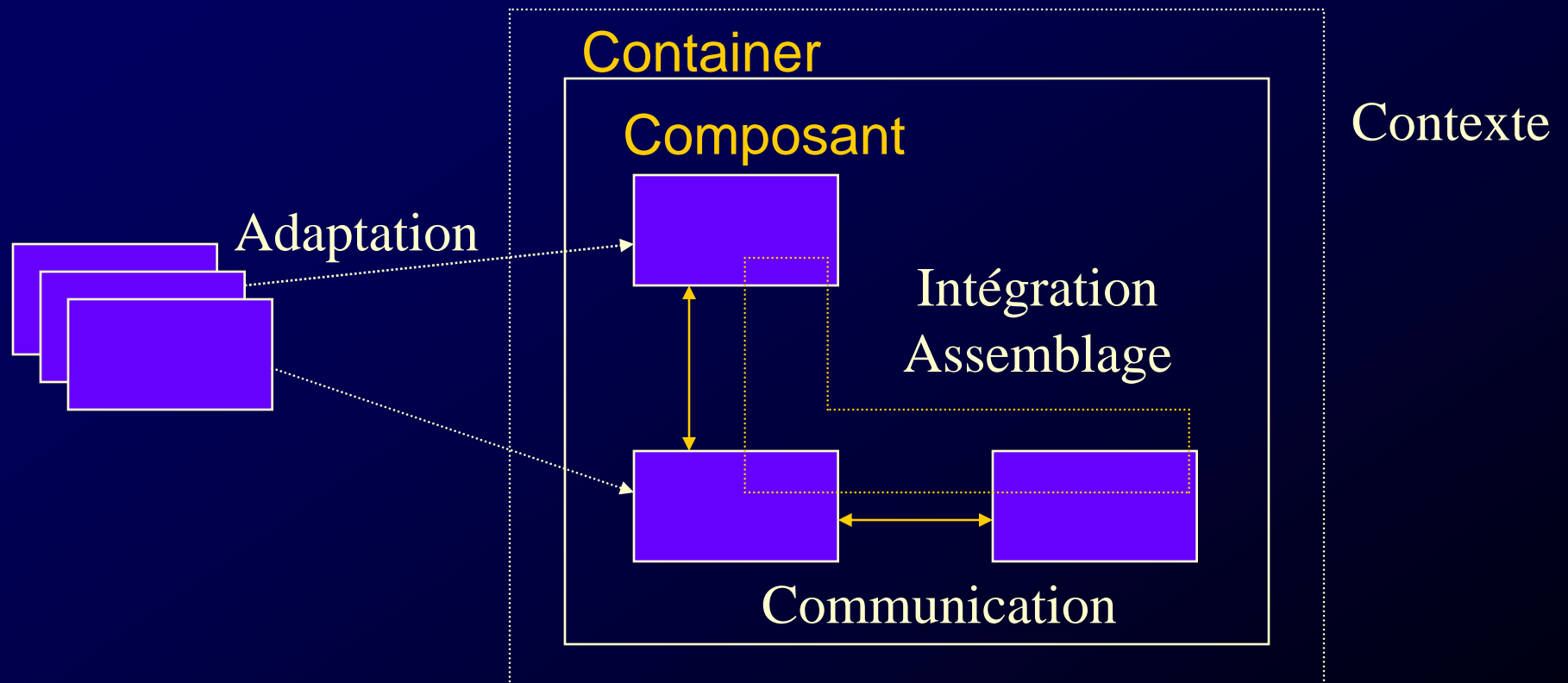
---

- ◆ Deux écoles pour la flexibilité
  - Composants flexibles, faciles à modifier et à adapter
    - White-Box
      - Frameworks: Squelette à partir duquel on construit l'application
    - Grande échelle
  - Composants spécialisés, où il n'y a rien à changer
    - Black-Box
    - Petite échelle



# Composants

---



# *Composants: classification*

---

- ◆ Structures de données / Types de Données Abstraits
  - Piles, Files, Chaînes, Ensembles, Listes, Arbres, Graphes
- ◆ Outils / Abstraction d'algorithmes
  - Filtres, Pipes, Ordonner, Rechercher, Comparer
- ◆ Élément GUI
  - boutons, fenêtres, ...
- ◆ Sous-Systèmes
  - Tout ce qui peut être construit

# JavaBeans

---

*« Un Bean Java est un composant logiciel réutilisable qui peut être manipulé visuellement dans un outil d'aide à la construction d'applications »*

# *JavaBeans*

---

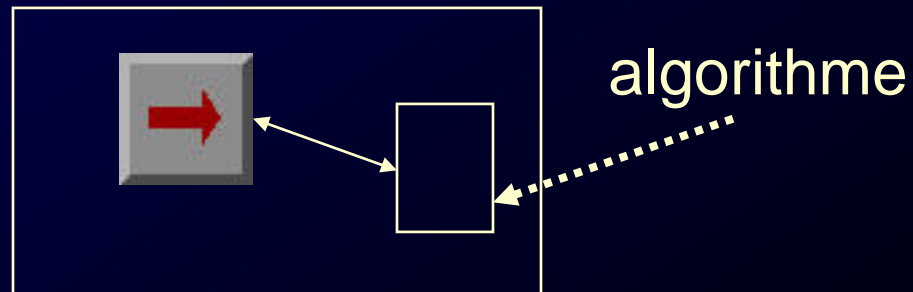
- ◆ But
  - Définir un modèle de composants logiciels dans lequel des composants Java peuvent être créés et composés dans des applications par des utilisateurs (programmeurs)
- ◆ Programmation Visuelle
  - Création de programmes par manipulation de la représentation graphique des composants
- ◆ Editeur de composition visuelle
  - Outil permettant de développer des programmes en arrangeant et connectant visuellement des « objets » software (beans)

# JavaBeans

---

## ◆ Types de Beans

- **Visuel**: Bean visible dans une interface graphique
  - bouton, texte, fenêtre
- **Non Visuel**: Bean invisible dans une interface graphique
  - algorithme de tri
  - mais ... un bean non visuel possède une représentation graphique qui permet de le manipuler visuellement dans l'outil



# *JavaBeans*

---

## ◆ La Double Vie des Beans

- Lors de la conception (design-time)
  - Le bean est manipulé dans l'outil d'aide à la construction
  - Le bean doit fournir des informations sur son apparence et son comportement et permettre des adaptations
- Lors de l'exécution (run-time)
  - Le bean est utilisé dans une application en cours d'exécution
  - Certaines méthodes utilisées pendant le design-time ne seront plus utilisées (ni même fournies) pendant le run-time

# JavaBeans

---

- ◆ Quelle est la classe d'un Bean ?
  - N'importe quelle classe!!
    - Si le bean est visible, il peut être sous-classe de `java.awt.Component`
    - Si le bean est invisible, il peut être de n'importe quelle classe
- ◆ Quelle interface un Bean doit-il implémenter?
  - Aucune en particulier!!
  - Mais, ... si on veut qu'il soit persistant, il faut implémenter:
    - Interface `Serializable`

# *JavaBeans: Caractéristiques*

---

- ◆ Pour qu'un outil d'aide à la construction d'applications puisse manipuler un bean, celui-ci doit fournir des caractéristiques: Propriétés, Événements, Méthodes
- ◆ **Propriétés**
  - Noms d'attributs associés à un bean, qui peuvent être lus ou écrits en appelant des méthodes appropriées.
  - L'outil permet de choisir la valeur des propriétés lors de la conception



# *JavaBeans: Caractéristiques*

---

## ♦ Événements

- Permettent la communication entre beans.
- Un bean *source* informe un bean *listener* qu'un certain événement a eu lieu

## ♦ Méthodes

- Méthodes publiques

# JavaBeans: Introspection

---

- ◆ L'outil de construction d'applications doit pouvoir analyser le bean (connaître les propriétés, événements, et méthodes)
- ◆ Deux approches:
  - Classe Supplémentaire
    - Définition explicite des caractéristiques disponibles
    - Définir une classe supplémentaire (BeanInfo Interface)

```
class MonComposant {}  
class MonComposantBeanInfo implements BeanInfo {}  
java.lang.reflect
```
  - Conventions
    - Conventions pour écrire des méthodes et interfaces servant à la détermination et adaptation des Propriétés, Événements, Méthodes

# JavaBeans: outils

---

- ◆ Applications de développement qui manipulent des Beans:
  - VisualAge for Java
  - BeanBox, fourni avec Bean Development Kit (BDK)
  - *java.beans package*  
(événement, sérialisation, introspection)
- ◆ Livraison d'un bean
  - Classe implémentée ou
  - Chablon sérialisé
- ◆ Instanciation d'un bean
  - A partir d'une classe (**new** ...)
  - A partir d'une sérialisation (**désérialiser** et **créer** l'instance)

# *JavaBeans: Sérialisation*

---

- ◆ Bean persistant:
  - sauver les propriétés, champs, états pour les retrouver plus tard
- ◆ Mécanisme disponible pour sauver: **sérialisation**
  - convertir le bean en une suite de données, écrites et stockées (dans un fichier)
- ◆ Pour retrouver le bean: **désérialisation**
  - reconstituer le bean à partir de la séquence de données stockées
- ◆ Interface Serializable

# *Beans ou Librairies?*

---

- ◆ Beans:
  - abstraction d'une entité conceptuelle
  - composants logiciels manipulés visuellement et adaptable
- ◆ Classes de librairies:
  - fournissent de la fonctionnalité à un programmeur
  - ne peuvent être manipulées visuellement

# *Autres Composants*

---

- ◆ Enterprise Java Beans (EJB, Sun)
  - Abstraction indépendante de l'implémentation dans le cadre des technologies d'entreprises
- ◆ Component Object Model (COM, DCOM, Microsoft)
  - Architecture logicielle qui permet à des applications d'être construites à partir de composants binaires
  - DCOM (Distributed COM): protocole permettant aux composants de communiquer de manière fiable, sûre et efficace à travers un réseau
- ◆ COTS (Components-off-the-shelf)
  - composant que l'on peut acheter, prêt à l'emploi, dans le « rayon virtuel » d'un fabricant

# *Une nouvelle entreprise*

---

- ◆ Est-ce qu'une petite entreprise peut survivre sur la vente de composants logiciels ?
  - Un composant est une classe (ou ensemble de classes)
    - Avec la documentation correspondante !
  - Les librairies Java (JDK) montrent qu'une clientèle pour des composants pré-développés existe
- ◆ Les éléments en faveur de l'entreprise
  - *1. Elle est reliée aux clients via l'Internet*
    - Et le nombre de clients potentiels est gigantesque
    - Elle livre ses marchandises directement aux clients
      - Frais de transport minimaux !

# *Succès de l'entreprise*

---

- *2. Les clients (des programmeurs) savent que le développement coûte*
  - Il est plus sage d'acheter un composant que de le faire soi-même
  - Un composant est complexe et on ne possède ni le temps ni le savoir-faire pour le développer
  - Un composant est un standard, et on est obligé de l'utiliser
    - P.ex: le code pour le chiffrement, les outils systèmes de communication
  - La programmation d'un composant non-erronné reste une tâche difficile



# *Succès de l'entreprise*

---

- 3. *Une classe est liée dynamiquement aux applications*
  - Un programmeur n'a pas besoin d'avoir tout le code à la compilation ou à l'installation
  - Parmi ses clients se trouvent tous ceux qui font de la maintenance et la mise à jour des applications !!!
- 4. *Les clients peuvent espérer un niveau supérieur de*
  - *Fiabilité* : les composants sont utilisés, et donc testés, plus souvent
  - *Compatibilité* : assurée par la présence d'interfaces standards dans le système

# *Succès de l'entreprise ...*

---

- *5. On aide le client à faire du prototypage rapide (rapid prototyping)*
  - Le but du client est toujours de développer le plus vite possible
  - Mais parfois la spécification n'est pas claire ; le client ne sait pas à 100% ce qu'il veut
    - Alors on construit une première version pour voir si elle marche et puis on la modifie petit à petit
      - La programmation à base de composants facilite le prototypage rapide puisque les composants sont encapsulés

## *.. mais les problèmes surviennent !*

---

- ◆ Après un moment, la gestion du rayon des composants commence à poser des problèmes
  - Elle dépend trop des notions de classe et d'objet !
- ◆ *1. Trop de redondance dans le code*
  - P.ex: les classes Carré et Cercle possèdent 80% de code en commun !
    - Trop de temps perdu dans le développement
    - Rappel: le but dans le développement est de réutiliser l'expérience ainsi que le code (pour le fournisseur aussi !)

# *Les problèmes de l'entreprise*

---

- ◆ 2. *Un composant ne satisfait pas les besoins à 100%*
  - Un cercle possédant une couleur ne peut pas être représenté par notre classe Cercle
  - Un document dans une bibliothèque peut être représenté par une classe Document
    - Les messages comprennent date(), texte()
    - Mais désormais, un document est multimédia
      - Il contient du son, des images,
      - Comment peut-on adapter les classes comme Document ?

# *Les problèmes de l'entreprise*

---

## ♦ *3. Recherche de composants*

- Comment retrouver le composant qu'on cherche dans un vaste rayon de classes ?
  - Parfois on recherche un ensemble de classes pour une tâche (un framework ou un pattern)
  - On peut chercher une classe à partir de son interface ou d'un descriptif textuel de son comportement
    - Ces besoins nécessitent l'utilisation d'une base de données avancée
- Et lorsqu'on trouve une classe, comment vérifier qu'elle correspond exactement aux besoins du programmeur ?

# *Les problèmes de l'entreprise*

---

- ◆ D'où, ... la nécessité de gérer les composants
  - département gestion des composants
    - département à part entière qui promeut l'utilisation des composants, et qui gère les composants et leur construction
  - système de composants
    - librairie de composants
    - partager les composants entre plusieurs projets
  - construction des composants
    - interface (bien définie, bon choix des noms)
    - indépendance de l'application
    - abstraction générale

# *Les problèmes de l'entreprise*

---

- ◆ 4. *Protection de droits de vente (le piratage)*
  - Même si l'entrepreneur demande un paiement pour ses composants
    - Ses composants peuvent être facilement réutilisés ou copiés sans que l'entreprise ne reçoive un paiement
  - Ce problème nécessite des solutions techniques ainsi que juridiques