Introduction à Java

Aperçue du cours

- Objectifs
  - Avoir une vision relativement exhaustive de la technologie.
  - Acquérir les bons réflexes lors de la conception de projets.
  - Implémenter correctement le code source.
  - Utiliser les fonctionnalités avancées, et raccourcis, du langage.
  - Travailler sur des architectures types de production.

- Vue d'ensemble du cours
  - Introduction, Rappels (Concepts Objets, JavaBeans, Patterns, …), API, …
  - Exemples simples (déroulement de séquence, application basique, …)
  - Structuration de données (JDBC, XML, Reflection, Gui, …).
  - Environnement Web (Tomcat, Servlet, JSP, JSTL, Struts, …).
  - J2EE et EJB (Entity, Session, Message, XML, …).
  - Agents Mobiles et architectures distribués.
  - Systèmes embarqués (MIDP, CLDC, Personnal Java, JavaCard, JavaRing, …).

Introduction

- Comparaison
  - *Code natif* : 2 tiers (architecture physique, code exécutable)
  - *ByteCode*: 3 tiers (architecture physique, machine virtuelle, code interprété)

- Composants du JDK:
  - Librairies (Bibliothèques de composants réutilisables)
  - Compilateur : « javac » (Conversion code source / byte-code)
  - Interpréteur : « java » (« Exécution » du byte-code)
  - AppletViewer (Visualisateur d'applet Web)
  - Bridge Natif (Intégration de code natif : dll)
  - Archives (Gestion sous la forme de jar utilisant un algorithme ZIP)
  - Clefs (Génération de clé permettant de signer des archives)
  - Versions (Gestion du versionning)

- Les environnements:

J2SE	- JavaBeans (modèle de composants) - Applets (embarquées dans une page Web) - Applications (lancées par une commande système)
J2EE	- Servlets, JSP, JSTL et TagLibs (composants Web) - EJB (composants distribués)
J2ME	- MIDP/CDLC (applications pour téléphone et *SmallDevice)* - JavaCard (applications pour smartcard) - Personnal Java (applications pour les PDA)

Grammaire et éléments du langage :

La Syntaxe

- Une grammaire identique au C (opérateurs *for, while, break continue, if - else, switch - case*).
- Les Primitives : des données atomiques se différenciant des Objects (*short, int, float, ...*)
- Les tableaux en Java sont des objets à part entière.
- Gestion des exceptions : (opérateurs *try - catch – finally, throws)*.

Les opérateurs particuliers

- « *this* » : Obtenir un descripteur sur l'objet courant.
- « *super* » : Atteindre la classe père d'un composant
- « *null* » : Permet de tester l'existence d'un objet.
- « *instanceof* » : Vérifie la cohérence de type pour un objet.

Les Concepts

- *Class* : Un ensemble de champs et de méthodes
- *Interfaces* : Un contrat de spécification de champs et de méthodes pour les classes.
- *Object*: Des instances de Classes
- *Constructor* : Opérateur « *new* » et méthodes spécifiques.
- *Modifiers* : Attributs « *public* » « *protected* » et « *private* »
- *Abstract Class*: Un modèle intermédiaire entre les Classes et les Interfaces

Les Mécanismes et principes Objet.

- L'*héritage*: Différence notable par rapport au C++.
- L'*encapsulation* : Couche logique d'accès aux données.
- Le *Polymorphisme* : Gestion de plusieurs types pour un objet donné.
- L'*Override* [1]: La *Redéfinition* (réécriture) de méthodes héritées.
- L'*Overload* : La *Surcharge*: services semblables, mais signatures de méthodes différentes
- Le lien dynamique : Accès automatique de méthodes redéfinies dans les classes filles.
- Le *Cast* Transtypage d'un objet selon un type donné. (*throws exception*)

La Sécurité

- Gestion d'exceptions : (*try - catch – finally*), Prédiction par déclaration de throws.
- Model de sécurité restreint sur les applets en JDK 1.0 et 1.1 (« *SandBox* »)
- Model étendue, ayant une granularité forte, à partir du JDK 1.2.
- ConstantPool, Décompilation et Obfuscation.

Les Composants Graphiques

- AWT : modèle 2 tiers
- JFC et Swing : modèle 3 tiers (pattern **M**odel **V**iew **C**ontroler)
- Evènements modèle « producteur-consommateur » (*Event* et *Listener*)
- Layouts : responsables du positionnement des composants.

[1] Souvent surcharge et redéfinition sont confondus…

Packages importants de l'API :

- *java.applet*

  Rôle : Dédié à l'environnement d'un client Web sous la forme d'une applet Java
  Classes majeures : *Applet, AppletContext*
  Fonctionnalité :
  * Contexte du navigateur,
  * Accès aux ressources serveur *(getImage(URL url), getAudioClip(URL url), ...)*
  * Lecture de Méta-Informations *(getCodeBase(), getDocumentBase(), getLocale())*

- *java.awt*

  Rôle : fournir une couche pour les Composants Graphiques de 1ère génération, (JDK 1.0 et 1.1).
  Classes majeures : *Layouts, Color, Component, Event, Font, Graphics, Image, Panel,*
  Fonctionnalité :
  * Primitives d'objets graphiques *(Button, List, Checkbox, TextField, TextArea, , ...)*
  * Containers haut niveau *(Component, Panel, Windows, Dialog, ...)*
  * Responsables de positionnement interne *(GridLayout, BorderLayout, ...)*
  * Objets graphiques de bas niveau *(Rectangle, Point, Polygon, Color, ...)*
  * Objets graphiques de haut niveau *(Toolkit, Graphics, PrintJob, ...)*
  * Evènement et Exceptions spécifiques *(Event, EventQueue, AWTException, ...)*
  Sous-packages :
  * *color, datatransfer, dnd, event, font, geom, im, image, print*

- *java.beans*

  Rôle : Définir et accéder les propriétés communes d'objets Java.
  Classes majeures : *BeanDescriptor, Beans, Encoder, Introspector, XMLDecoder, XMLEncoder*
  Interfaces majeures : *BeanInfo, ExceptionListener, PropertyChangeListener, Visibility*
  Fonctionnalité :
  * Introspection d'objets.
  * Persistence d'objects par sérialization vers un médium cible (file, socket, DB, …),
  * Model évènementiel fort pour l'administration d'état d'objets.
  Sous-packages :
  * *beancontext*

- *java.io*

  Rôle : Fournir des objets permettant la manipulation de canaux générique d'entrée-sortie.
  Classes majeures : *File, InputStream, OutputStream, PrintStream, Reader, Writer*
  Interfaces majeures : *DataInput, DataOutput, ObjectInput, ObjectOutput, Serializable*
  Fonctionnalité :
  * Descripteurs générique de Flux d'Entrée-Sortie (*InputStream, OutputStream)*
  * Différents canaux mis à disposition *(Buffer, Byte, Char, Data, File, LineNumber, Object, ...)*
  * Objets de haut niveaux pour la lecture et l'écriture (*Reader, Writer)*

- *java.math*

  Rôle : Dépasser les limites de valeurs imposées par les primitives.
  Classes majeures : *BigDecimal, BigInteger*

- *java.lang*

  Rôle : Met en place les classes propres au noyau du langage.
  Classes majeures : *Number, String, Object, System*
  Interfaces majeures : *Cloneable, Comparable, Runnable*
  Fonctionnalité :
  * Classes de base du langages *(Class, Object, Package, Void)*
  * Classes fonctionnelles basiques *(Math, String, StringBuffer, ...)*
  * Module de la Machine Virtuelle *(ClassLoader, Compiler, Runtime, SecurityManager, ...)*
  * Abstractions (*System, Thread, Process, Throwable, ...)*
  * Mapping des primitives en objet *(Boolean, Byte, Character, Double, Float, Integer, Long, Short)*
  * Exceptions principales *(Exception, RuntimeException)*
  * Sous-Exceptions *(NullPointerException, ClassCastException, NumberFormatException, ...)*
  * Erreur principale *(Error)*
  * Sous-Erreurs *(OutOfMemoryError, InternalError, UnknownError, ...)*
  Sous-packages :
  * reflect : Classes de manipulation génériques du noyau objet du langage.
  Elles permettent d'atteindre des ressources de classes *(Field, Method, Constructor, Modifier, ...)*

- *java.net*

  Rôle : Permettre les fonctionnalités usuelles afférentes au Réseau et a ses protocoles classiques.
  Classes majeures : *InetAddress, ServerSocket, Socket, URL,*
  Interfaces majeures : *ContentHandlerFactory, SocketImplFactory, SocketOptions,*
  Exceptions majeures : *BindException, ConnectException, ProtocolException, SocketException,...*
  Fonctionnalité :
  * Client TCP/UDP (*DatagramSocket*, *Socket, MulticastSocket, ...)*
  * Serveur *(ServerSocket)*
  * Manipulation de ressources *(URI, URL, HttpURLConnection, ...)*
  * Gestion de droit *(Authenticator, NetPermission, PasswordAuthentication, SocketPermission, ...)*
  * Encodage/décodage *(URLDecoder, URLEncoder)*
  * Gestion d'adresse IP dans différentes versions *(InetAddress, Inet4Address, Inet6Address)*

- *java.nio*

  Rôle : Mise à disposition de *Buffer* pour les entrées/sorties.
  Classes majeures : *Buffer*
  Fonctionnalité :
  * Buffer dédiés aux types des primitives *(ByteBuffer, CharBuffer, DoubleBuffer, FloatBuffer, ...)*
  Sous-packages :
  * *channels* : Connection vers des ressources de type entrée/sortie,
  permet aussi de définir des *Selector* (opération I/O non bloquante multiplexées)
  * *charset* : fourni des *Encoder* et *Decoder* pour les conversion de caractère byte/Unicode.
  * *channels.spi / charset.spi* : classes permettant l'**I**mplémentation par des **S**ervice **P**rovider.

- *java.rmi*

  Rôle : Permet l'invocation de méthodes distantes à travers le réseau.
  Classes majeures : *MarshalledObject, Naming, RMISecurityManager*
  Interfaces majeures : *Remote*
  Sous-packages : *activation, dgc, registry, server*

- *java.security*

  Rôle : Fourni les classes et interfaces propre au Framework de Sécurité du langage
  Classes majeures : *AccessControler, KeyFactory, MessageDigest, Permission, Signature, Signer*
  Interfaces majeures : *Certificate, Key, Principal, PrivateKey, PublicKey*
  Fonctionnalité : Vérification et manipulation diverses dans le cadre de sécurité de la VM.
  Sous-packages : *acl, cert, interfaces, spec*

- *java.sql*

  Rôle : Framework permettant d'accéder des données issues de *DataSource* (SGBD, …)
  Classes majeures : *Date, DriverManager, Time,*
  Interfaces majeures : *Connection, Driver, ResultSet, Statement*
  Fonctionnalité :
  * Manipulation de types SQL *(Blob, Clob, Struct)*
  * Fonctionnalité avancées SQL *(CallableStatement, PreparedStatement, ...)*.
  * Méta-Informations *(DatabaseMetaData, ParameterMetaData, ResultSetMetaData, ...)*

- *java.text*

  Rôle : Manipulation de textes, dates, nombres et messages, indépendamment du langage naturel.
  Classe majeures : *Bidi, BreakIterator, Collator, DateFormat, DecimalFormat, NumberFormat*
  Fonctionnalité :
  * permet d'effectuer les manipulations usuelles sur des chaînes de caractères.

- *java.util*

  Rôle : Fournir des classes et interfaces utilitaires riches.
  Classe majeures : *Calendar, Collections, Date, HashMap, Locale, Random, Stack, Timer, Vector*
  Interfaces majeures : *Collection, Comparator, EventListener, Iterator, Map, Observer, Set*
  Fonctionnalité :
  * Structures simples de données *(HashTable, Vector, Stack, StringTokenizer, (JDK 1.0))*
  * Enrichie par le Framework Collection *(Map, Set, List, Iterator, ...)*
  * Gestion d'évènements génériques *(EventListener, EventObject, ...)*
  * Internationalisation *(Locale, ResourceBundle, Properties, ...)*
  * Utilitaires pratiques *(Timer, Calendar, TimeZone, Random, ...)*
  Sous-packages : *jar, logging, pres, regex, zip*

- *javax :* Compléter le noyau des packages *java* par des packages d'eXtensions.

accessibility :	Gestion de fonctionnalité propres aux mécanismes d'assistance.
crypto :	Opérations de cryptographie.
imageio :	Manipulation avancée d'images.
naming, naming.ldap :	Gestion d'annuaire, localisation d'objets par nomenclature.
net, net.ssl :	Divers *Factory* de *socket* (client/serveur), protocole SSL.
print :	Gestion d'impression avancée
rmi, rmi.CORBA :	*PortableRemoteObject*, et *bridge* vers CORBA/IIOP.
security.auth, security.cert :	Authentification, Autorisations, Certificats, …
sound.midi, sound.sampled :	Gestion de l'interface Midi, et de processus de sampling
sql :	*PooledConnection, DataSource, RowSet, XAConnection.*
swing :	Refonte du Framework de composants graphique selon MVC.
transaction, transaction.xa :	Exceptions spécifiques, Transactions distribuées.
xml.parsers, xml.transform :	Parsers SAX et DOM, transformations associées.

- *org.ietf* : librairies, publiées par l'*Internet Engineering Task Force,* orientées réseau et sécurité.

  jgss : Api unifiée d'accès à des services de sécurité.

- *org.omg* : librairies, publiées par l'*Object Management Group*, dédiées à CORBA.

  CORBA : Mapping de la technologie sur le langage Java.
  CosNaming : Service de *naming* pour les IDL Java.
  Dynamic : Objects spécifiques aux spécifications de l'OMG.
  IOP : Module IOP des spécifications de CORBA.
  Messaging : Module de messages de CORBA.
  PortableInterceptor : Gestion des flux de l'ORB.
  PortableServer : Classes et Interfaces communes (haut niveau).
  stub.java.rmi : Stub RMI-IIOP pour le package java.rmi.

- *org.xml* : librairies de parser XML.

  sax : Classes et interfaces du parser SAX.
  sax.ext : Extensions de classes et d'interfaces pour le parser SAX2.
  sax.helpers : Classes « helpers » pour les processus SAX.

- Les API additionnelles fournie par Sun (source http://java.sun.com)

J2SE Optional Packages - Java Advanced Imaging 1.1 API Documentation - Java Communications 2.0 API Documentation - Java Cryptography Extension (JCE) 1.2.1 API Documentation - Java Data Objects (JDO) API Documentation - Java Naming and Directory Interface (JNDI) 1.2.1 Specification - Java Secure Socket Extension (JSSE) 1.0.2 API Specification - Java Speech 1.0 - Java 3D 1.2 API Documentation (javadoc) - Java Media Framework API Documentation **Java Telephony API (JTAPI)** - Java Telephony 1.4 Specification **Java 2 Platform, Enterprise Edition (J2EE)** - J2EE 1.4 Beta API Specification - J2EE 1.3.1 API Specification - J2EE 1.2.1 API Specification	**Java 2 Platform, Micro Edition (J2ME)** - CLDC 1.0 API Documentation (zip download) - JSR 66 RMI Optional Package - JSR 37 Mobile Information Device Profile Final Specification Release - JSR 36 Connected Device Configuration Specification - JSR 46 Foundation Profile Specification **Java Card** - Java Card 2.2 Platform Specification - Java Card 2.1.1 Platform Specification **XML Technologies** - Java Web Services Developer Pack 1.0 Combined API Specification **Other Technologies** - Java Embedded Server API Documentation (zip download) - Java TV 1.0 API Specification - JAIN API Specifications - JSR 03 Java Management Extensions (JMX) - JiniTM Network Technology - APIs and Specs - JXTA v1.0 Protocols Specification - OSS through Java API Specification and Reference

- Intégration d'API spécifiques externes :
  - Jasper : Modélisation avancée de l'impression.
  - Struts : Pattern MVC pour le Web.
  - Log4j : Gestion de logs.
  - WebSSO : Gestion de Session clientes.

Forums de discussion sur le langage Java :

http://forum.java.sun.com/index.jsp

General
- New To Java Technology
- Java Programming
- Java Programming [Archive]
- Adding Generics
- Advanced Language Topics
- Native Methods
- 100% Pure Java
- J2EE SDK
- Java Community Process Program
- Java 2 Software Development Kit
- Java BluePrints
- Java Game Development
- Java Desktop Application Development
- Java Errors and Error Handling
- UML, OO Design, Patterns
- Algorithms

Runtime Environment
- Java Runtime Environment
- Java Virtual Machine
- Java HotSpot

GUI Building
- Project Swing
- Project Swing [Archive]
- Abstract Window Toolkit (AWT)
- Java 3D
- JavaBeans
- Java Media Framework
- Java 2D
- Accessibility APIs
- Java Applet Development
- Java Event Handling

Essential Classes and Documentation Generation
- Java Collections Framework
- Internationalization
- Javadoc Tool
- JavaHelp
- Serialization

Majc
- New To MAJC
- Chip Level Multi Processing
- Wirespeed Computing

Distributed Computing
- Jini Network Technology
- The Brazil Project
- JDBC
- Enterprise JavaBeans
- JavaMail
- Interface Definition Language (IDL)
- Naming and Directory (JNDI)
- Remote Method Invocation (RMI)
- RMI-IIOP
- Serialization
- Java Transactions (JTA/JTS)
- Java Message Service (JMS)
- Distributed Computing General
- J2EE Patterns

Web Services and Applications
- Java Technologies for Web Services
- Java Technology & XML
- JavaServer Pages
- Java Servlet Technology
- JavaServer Faces Technology

Security
- Signed Applets
- Cryptography
- Security Managers
- Java Secure Socket Extension
- Security General

Installation and Compiling
- Compiling
- Installation

Debug and Deploy
- Java Archive (JAR) Files
- Java Extension Mechanism
- JDB Tool
- Java Plug-In
- JavaBeans ActiveX Bridge
- Java Web Start & JNLP

Sun Developer Network
- Discuss the Sun Developer Network Web Site
- New Forum Suggestions

Liens additionnels intéressant:

http://java.sun.com
http://www.ibm.alphaworks.com
http://www.developer.com
http://www.gamelan.com
http://developer.java.sun.com/developer/codesamples/examplets/index.html

Outils de Développements :

Sun : **Java One Studio** (Forte)

- Composants "data-aware" (dbSwing)
- Déploiement multi-plateforme.
- Génération de Servlets, JSP, Applets, JavaBeans, via des wizards.
- Intégration avancée avec le serveur Sun One Server.
- Version « Mobile » permettant de tester des applications J2ME (MIDP, CLDC).
- Intégration possible d'émulateur de téléphone Nokia, Sony-Ericsson, Siemens, Motorola, Palm.

Inprise : **Jbuilder**

- RAD Java 2 JFC/Swing, applets, JSP/Servlets, JavaBeans™,.
- Plus de 300 JavaBeans fournis avec leurs codes source.
- Visualisation et refactoring de fichiers XML.
- Support avancé des EJB.
- Runtime de Servlets et de JSP permettant de tester en locale.
- Wizard d'accès aux bases de données, de création de composants Web et d'EJB.
- Version Mobile permettant de développer des applications i-Mode
- Déploiement vers les environnements BEA WebLogic, IBM WebSphere, Sun ONE,® Oracle9i,™ Sybase® EAServer, JBoss,® et le serveur intégré Borland® Enterprise Server.

MetroWerks: **CodeWarrior for Java**

- Basé sur le noyau de l'IDE de codeWarrior.
- Test avancé pour des développements sur PDA pour chaque plateforme (ARM, SH3, ..).
- Intégration de la technologie JavaCard.
- Version Mobile pour les développements sur téléphones portables.

Oracle : **Jdeveloper**

- Forte Intégration avec l'environnement d'Oracle.
- Wizards sur la création de composant graphique et accès aux Bases.
- Support de Corba (Wizard pour le dialogue par IIOP).
- Librairies additionnelles intéressantes.
- Implémentation de l'infoBus pour l'échange de données pour les JavaBeans.

Microsoft : **Visual J++**

- Compilation native ! génération de *.exe* a partir de Java.
- Runtime dédié Windows, optimisé car lié aux librairies Système.
- Interface d'installation et d'extraction des projets.
- Bridge vers le standard COM : Intégration des objets et dll natives du Système.

IBM : **Eclipse**

- Outil free et openSource (développé par OTI, puis racheté par IBM).
- Modularité forte : développement ouvert de plug-in par la communauté eclipse (IDE : PDE).
- Permet de travailler sur différent types de fichier (C, HTML, Java, XML, JSP, GIF, …).
- SWT (Standard Widget Toolkit) de composants graphiques (émulation native, ou dynamique).
- Création du framework JFace, qui manipule de façon avancée les composants SWT.
- Intégration de l'outil Ant.

Outils additionnels :

- Test

  JUnit, Cactus, DBUnit.

- Déploiement

  Ant, Maven, Clover.

- Solutions Serveurs Web :

Containeur de Servlets	URL
Bluestone	http://www.bluestone.com
Borland Enterprise Server	http://www.inprise.com
iPlanet Application Server	http://www.sun.com/software/iplanet/
Orbix E2A (iPas)	http://www.iona.com
Jetty	http://www.mortbay.com
Jrun	http://www.macromedia.com/software/jrun/
Orion Application Server	http://www.orionserver.com
Resin	http://www.caucho.com
Silverstream	http://www.silverstream.com
Apache Tomcat	http://jakarta.apacha.org/tomcat/
Weblogic Application Server	http://www.bea.com
WebSphere	http://www-4.ibm.com/software/webservers/appserv/
EAServer	http://www.sybase.com

- Solution Serveur J2EE :

Container J2EE	Composants
Bea :	WebLogic + Tuxedo.
IBM :	WebSphere + Visual Age + Component Broker Connector.
Borland (Inprise):	Visibroker.
Netscape :	Netscape Application Server.
JBoss	Jboss, Junit, Xdoclet, Ant

Rappel sur les Design Pattern

Création	
Factory Method	Permet de dédier à une méthode spécifique, redéfinie dans des classes filles, la création d'objets.
Abstract Factory	Met en place une fabrique *abstraite* d'objets. A redéfinir ensuite dans les classes *concrètes* spécialisées.
Builder	Responsable de la création d'une structure spécifique pour un composant..
Prototype	Objet servant de modèle de données et de classe pour des processus standard type clonage (*Object.clone()*).
Singleton	Objet qui n'est présent qu'une seule fois dans une application. Dédié à être stocké dans des *Registry* de composants similaires.
Structuration	
Adapter	Permet de simplifier le traitement d'un composant en introduisant une classe intermédiaire.
Composite	Arborescence d'objets éventuellement de même type.
Proxy	Responsable de la gestion de ressource distante.
Flyweight	Allègement de classes, par externalisation de structures communes dans des fichiers. (singleton d'interface de constantes, par ex).
Façade	Encadrement d'un ensemble de traitements, souvent complexe, dans un objet spécifique.
Bridge	Segmentation de la logique et de son implémentation. (*stub/skeleton*, *EJBObject* et *Entity/Session/Message*).
Decorator	Ajout dynamique de fonctionnalité à un composant.
Comportement	
Chain of Responsability	Traitement par une chaîne de la consommation d'un objet.
Command	Abstraction d'une action.
Interpreter	Responsable de l'interprétation de données complexe.
Iterator	Abstraction du mécanisme d'itération. Ce pattern est une classe officielle du framework Collection de java.
Mediator	Intermédiaire entre 2 composants n'ayant pas relation directe. Il permet de gérer les échange de références entre objet, et donc leur logique.
Observer	Objet dédié à l'étude de l'état d'un objet, et à l'invocation d'appels selon son évolution..
State	Etat dans lequel se trouve un composant. Par exemple MVC.
Strategy	Gestion de différentes solutions possible de traitement pour un cas donné.
Template Method	Abstraction d'un modèle de méthode redéfini dans les arborescences de classes.
Visitor	Mécanisme en 2 temps d'inversion d'un traitement logique d'un composant vers un *Visitor*.

JavaBeans

Les spécifications sur les JavaBeans recommandent certains principes lors de développement :

- Le respect de l'encapsulation, une des règles des langages orientés objets.
  L'accès des champs internes des objets ne peut pas se faire directement vers les champs eux-mêmes (déclarés en *private*), mais par le biais de méthodes (*public*).

  Les *Accessor* (méthodes *get, set, is* utilisant le nom du champ) permettent les lectures/écritures des valeurs au sein des objets :

```
public class simpleObject
{
 //déclaration des champs privés

 private String      description;
 private int         values[];
 private boolean     flagOK;

 //constructeurs

 public simpleObject() {}
 public simpleObject(String description)
 {
  setDescription(description);
 }

 //Accessor de champ simple

 public String getDescription()
 {
  return description;
 }
 public void setDescription(String newDesc)
 {
  description = newDesc;
 }

 //Accessor de champ booléen

 public boolean isFlagOk()
 {
  return flagOK;
 }
 public void setFlagOK(boolean newFlagOK)
 {
  flagOK = newFlagOK;
 }
```

```
 //Accessor de champs indexés

 public int[] getValues()
 {
  return values;
 }
 public void setValues(int[] newValues)
 {
  values = newValues;
 }

 public int getValues(int index)
 {
  return values[index];
 }
 public void setValues(int newValue, int index)
 {
  values[index] = newValue;
 }
}
```

- L'application de cette norme permet d'utiliser des mécanismes d'*introspection*.
  Ces processus puissants donnent un accès directe aux membres public des classes :
  Définition, constructeurs, méthodes, champs et droits.

  Exemples d'utilisations usuelles de l'API de réflexivité (package *java.lang.reflect*) :

```
try
{
  //Accès aux ressources de classes :

  Class objectClass        = Class.forName(strClassName);
  // Class objectClass     = myObject.getClass(); //accès possible depuis n'importe quel objet

  Field objectField        = objectClass.getField(strFieldName); //ce champ doit être déclaré public

  Class tabParameterTypes[] = {};  //la méthode doit être déclarée public, ici une méthode getX()
  Method getMethod         = objectClass.getMethod(strMethodName, tabParameterTypes);

  //recherche d'un constructeur basique, sans argument.
  Constructor classConstructor = objectClass.getDeclaredConstructor(tabClassParameterType);


  //Utilisation des ressources:

  //lecture d'un champ, nécessite un objet cible dans lequel lire le champ
  Object fieldValue        = objectField.get(objectTarget);

  //affectation d'un champ
  objectField.set(objectTarget, value);

  //invocation d'une méthode get, la méthode ne prend pas de paramètre en entrée
  Object tabObject[]       = {};
  Object objectResult      = getMethod.invoke(objectTarget, tabObject);

  //invocation du constructeur pour instancier un objet à la volée.
  Object objectCreated     = classConstructor.newInstance(tabObject);
}
catch (Exception e) {System.out.println(e.toString());}
```

- Les spécifications de JavaBeans mettent aussi en exergue la *persistance* des objets.

Elle permet d'utiliser par des médias physiques des objets, par exemple stocker sur le File System des objets, ou bien les envoyer à travers le réseau par des Sockets, ou bien selon des protocoles gérant la partie transmission (CORBA, RMI, par exemple).

La persistance passe par le biais de l'interface *java.io.Serializable*, qui permet à la machine virtuelle d'exploiter l'objet en tant que données brutes. On peut, par exemple, sérialiser vers des fichiers, des données de Session d'un client Web, puis les recharger sous leur forme objet en les relisant lors de la reprise de Session de ce client.

- Les JavaBeans fournissent une classe permettant le traitement d'événement génériques selon le modèle bien connu « producteur-consommateur ». Ce concept permet d'automatiser la communications entre objets au sein d'une application.

```
import java.util.EventListener;

public interface UpdateListener extends EventListener
{
  public void updateInfo(UpdateEvent e)
}
```

```
public class UpdateEvent
{
  private String newDescription;

  public updateEvent(String newDescription) {setNewDescription(newDescription);}

  public String getNewDescription()  {return newDescription;}
  public void setDescription(String newDescription) {this.newDescription = newDescription;}
}
```

```
public class myBean implements UpdateListener
{
  private String      description;

  public String getDescription()   {return description;}
  public void setDescription(String newDesc) {description = newDesc;}

  public void updateInfo(UpdateEvent e)
  {
    setDescription(e.getNewDescription());
  }
}
```

```
import java.util.Vector;

public beanRegister
{
  private Vector vectorListeners = new Vector();

  public void addUpdateListener(UpdateListener ul)   {vectorListeners.add(ul);}
  public void removeUpdateListener(UpdateListener ul)   {vectorListeners.remove(ul);}

  public void overallUpdate(myBean sourceBean)
  {
    UpdateEvent e = new UpdateEvent(sourceBean.getDescription);

    for (int i = 0; i != vectorListeners.size(); i++)
      ((UpdateListener) vectorListeners.elementAt(i)).updateInfo(e);
  }
}
```

- Les JavaBeans définissent un format de fichier de déploiement, les *.jar* (Java Archives), qui permettent d'empaqueter les classes et ressources d'applications (fichiers multi-médias, descripteur de déploiement XML, descripteur du jar avec possibilité de signer à l'aide de certificats X509 le contenu pour en sécuriser la transmission sur Internet.