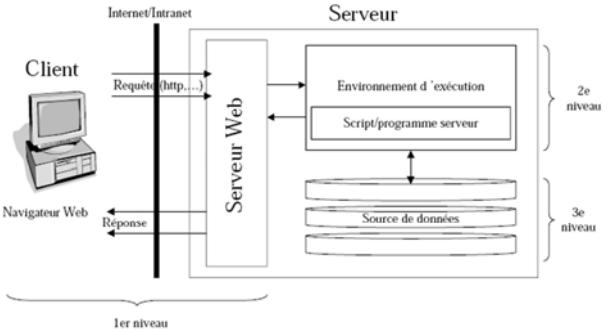
APPLICATIONS JAVA

JDBC
JSP
Servlet
MIDlet
Android

Applications Web

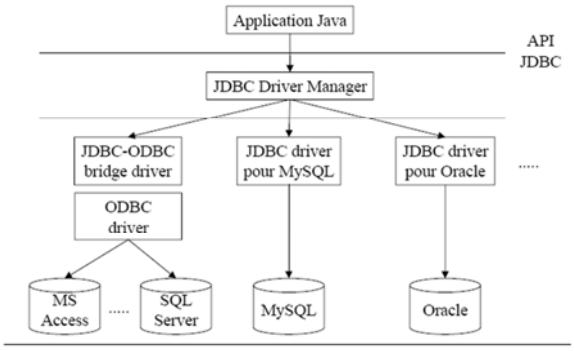


1. JDBC

Java Database Connectivity

JDBC - Java Database Connectivity

- Architecture d'une application Java-JDBC-SGBD



JDBC - Java Database Connectivity

- JDBC permet à un programme Java d'interagir
  - localement ou à distance
  - avec une base de données relationnelle
- Fonctionne selon le principe client-serveur
  - Un client : le programme Java
  - Un serveur : un SGBD (ex: MySQL)
- Principe
  - le programme Java charge le pilote
  - le programme Java ouvre une connexion
  - le programme Java envoie des requêtes SQL
  - le programme Java récupère les résultats
  - le programme Java effectue des traitement des données ...
  - le programme Java ferme la connexion

JDBC - Java Database Connectivity

- Connexion à une base de données
  - Une application Java doit mentionner l'URL de la une base de données :
    ```
    String NomUrl = "jdbc:SousProtocole:SourceDeDonnées";
    ```
  - Pour accéder à une source de données, il est nécessaire de disposer d'un pilote JDBC propre au modèle de la base de données.
    ```
    jdbc.NomDriver
    ```
  - Le *driver* doit être instancier et enregistre par une instruction spécifique : Pour utiliser le pilot JDBC-MySQL:
    ```
    Class.forName("com.mysql.jdbc.Driver");
    ```
  - Dès qu'un driver a reconnu le gestionnaire de base de données correspondant à l'URL fournie, il lance une connexion à la base et utilise le nom d'utilisateur et le mot de passe indiqués.
    ```
    Connection con = DriverManager.getConnection
                (Url, "Utilisateur", "MotDePasse");
    ```

JDBC - Java Database Connectivity

- Les requêtes de sélection :
  - L'objet *Connection* créé va permettre d'interagir avec la base. Pour réaliser des requêtes de sélection, un objet de type *Statement* doit être généré.
  - *Statement* symbolise une *instrution SQL*.
    ```
    Statement requete = con.createStatement();
    ```
  - Le résultat d'une requête est récupéré par un objet de type *ResultSet* et permet d'accéder aux données extraites grâce à la requête.
    ```
    ResultSet resultat = requete.executeQuery
                         ("select * from destinations");
    ```
  - Après la requête, le "curseur" est positionné juste avant la première ligne du résultat, la méthode *next()* permet d'avancer d'enregistrements en enregistrements séquentiellement : `resultat.next()`
  - Pour récupérer les données dans chaque colonne, l'interface *ResultSet* propose plusieurs méthodes adaptées aux types des données récupérées : `getString(NumCol), getInt(NumCol), getDate(NumCol)`

JDBC - Java Database Connectivity

- Les requêtes de mises à jour :
  - La mise à jour d'une base de données peut être effectuée par le biais d'une requête SQL de type *UPDATE*, *INSERT* ou *DELETE* à partir de la méthode *executeUpdate("Requête")* sur un objet *Statement*.
  - Le résultat renvoyé par l'exécution de la requête indiquera le nombre de lignes mises à jour dans la base, contrairement à une requête de sélection qui renvoie un *ResultSet*.
- Déconnexion :
  - La méthode *close()* permet de libérer les ressources prises par la création d'objets de type *ResultSet*, *Statement*, et *Connection*.
  - Elle existe pour chacune de ces interfaces. Elle est le plus souvent employée pour une *Connection*, car elle libère en même temps toutes les ressources qui lui sont associées.

JDBC - Java Database Connectivity

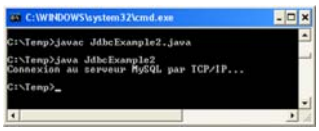
- Architecture d'une application JDBC

```
import java.sql.DriverManager;        // gestion des pilotes
import java.sql.Connection;           // une connexion à la BD
import java.sql.Statement;            // une instruction
import java.sql.ResultSet;            // un résultat (lignes/colonnes)
import java.sql.SQLException;         // une erreur

public class exempleJDBC {
   try {
        // chargement du pilote

        // ouverture de connexion

        // exécution d'une requête

        // Traitement des résultats

        // Fermeture de la connexion
   } catch (Exception ex) {  }
}
```


JDBC - Java Database Connectivity

- Connexion à une base de données :

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
public class JdbcExample2 {
  public static void main(String args[]) {
    Connection con = null;
    try {
      Class.forName("com.mysql.jdbc.Driver");
      con = DriverManager.getConnection
            ("jdbc:mysql://localhost:3306/hotel","user","123456");
      if(!con.isClosed())
        System.out.println("Connexion au serveur MySQL par TCP/IP...");
    } catch(Exception e) {
        System.err.println("Exception: " + e.getMessage());
    } finally {
        try {
        if (con != null)
          con.close();
        } catch(SQLException e) {}
    }
  }
}
```



JDBC - Java Database Connectivity

- Déclaration du pilote JDBC
  - Méthode de chargement explicite d'un pilote :

```
void loadDriver() throws ClassNotFoundException {
    Class.forName("com.mysql.jdbc.Driver");
}
```

  - L'appel à *forName* déclenche un chargement dynamique du pilote.
    - `Class.forName(String className)` : Retourne la classe d'objet associé à la classe ou l'interface donné avec le nom de la chaîne de caractères.
  - Un programme peut utiliser plusieurs pilotes, un pour chaque base de données.
  - Le pilote doit être accessible à partir de la variable d'environnement CLASSPATH.

JDBC - Java Database Connectivity

- Connexion à la base de données
  - Méthode d'ouverture d'une nouvelle connexion :

```
Connection newConnection() throws SQLException {

  final String url = "jdbc:mysql://localhost/test";
  Connection conn = DriverManager.getConnection(url,"user","pass");

  return conn;

}
```

  - L'URL est de la forme :
    jdbc:sous-protocole:sous-nom

JDBC - Java Database Connectivity

- Interroger une base de données
  - Méthode de composition d'une requête *SQL* de type *SELECT* :

```
// Définir la requête dans une chaîne de caractères
String query = "SELECT nom,prenom,age FROM etudiant";
// Envoi de la requête et récupération du résultat
ResultSet rs = st.executeQuery(query);
// Traitement des résultats
while (rs.next()) {
// Instruction qui récupèrent les données des champs-attributs
// suivant le nombre d'enregistrements
      rs.getString("nom_champs"|numéro);
      rs.getInt("nom_champs"|numéro);
      rs.getFloat("nom_champs"|numéro),
}
```


JDBC - Java Database Connectivity

- Les requêtes en JDBC

```
public void listPersons() throws SQLException {
    Connection conn = null;
    try {
        // créer une nouvelle connexion
        conn = newConnection();
        Statement st = conn.createStatement();
        // Définir, envoi de la requête et réception du résultat
        String query = "SELECT nom,prenom,age FROM etudiant";
        ResultSet rs = st.executeQuery(query);
         // Traitement des résultats
        while (rs.next()) {
            System.out.println(         // Récupération du contenu
                rs.getString("nom"),    // de l'attribut 'nom'
                rs.getString("prenom"), // idem 'prenom'
                rs.getInt("age") );     // idem 'age'
        }
    } finally {        // Fermer la connexion
        if (conn != null) conn.close();
    }
}
```


JDBC - Java Database Connectivity

- Modification de la base de données
  - Création d'une table dans la BD :
    une table "*personne*" avec trois attributs et une clé primaire non nulle et auto-incrémentée.

```
Statement st = conn.createStatement();
String query = "CREATE TABLE personne
               (id int not null auto_increment,
               prenom VARCHAR(15),
               nom varchar(15),
               age INT,
               primary key(id))";
st.executeUpdate(query);
```


JDBC - Java Database Connectivity

- Modification de la base
  - Insertion de lignes

```
Statement st = conn.createStatement();
int nb = st.executeUpdate(
    "INSERT INTO personne(Nom,Age) " +
    "VALUES ('" + nom + "', " + age + ")"
    );
```

  - Ce principe est aussi utilisable pour les instructions UPDATE et DELETE.

```
Statement st = conn.createStatement();
int nb = st.executeUpdate(
    "UPDATE personne " +
    "SET Age = " + age + " " +
    "WHERE Nom = '" + nom + "' "
    );
```


JDBC - Java Database Connectivity

- Correspondance des types Java / SQL et la correspondance des dates et heures :

SQL	Java
CHAR	String
VARCHAR	String
LONGVARCHAR	String
NUMERIC	java.math.BigDecimal
DECIMAL	java.math.BigDecimal
BIT	boolean
TINYINT	byte
SMALLINT	short
INTEGER	int
BIGINT	long
REAL	float
FLOAT	double
DOUBLE	double
BINARY	byte[]
VARBINARY	byte[]
LONGVARBINARY	byte[]

SQL	Java	Explication
DATE	java.sql.Date	codage de la date
TIME	java.sql.Time	codage de l'heure
TIMESTAMP	java.sql.TimeStamp	codage de la date et de l'heure

JDBC - Java Database Connectivity

```
import java.sql.*;
public class ExempleJdbc {          // Connexion à une BD MySQL avec JDBC et requête SQL
  public ExempleJdbc() {
    try {
         this.loadDriver();  this.listPersons();
    } catch (ClassNotFoundException e) {
         System.err.println("Pilote JDBC introuvable : " + e.getMessage());
    } catch (SQLException e) { }
  }
  void loadDriver() throws ClassNotFoundException {
    Class.forName("com.mysql.jdbc.Driver");
  }
  Connection newConnection() throws SQLException {
    final String url = "jdbc:mysql://localhost:3306/test";
    Connection conn = DriverManager.getConnection(url, "client", "123456");
    return conn;
  }
  public void listPersons() throws SQLException {
    Connection conn = null;
    try {
         conn = newConnection();   Statement st = conn.createStatement();
         String query = "SELECT nom,prenom,age FROM personne ORDER BY age";
         ResultSet rs = st.executeQuery(query);
         while (rs.next()) {
           System.out.println(rs.getString(1)+" "+rs.getString("prenom")+" "+rs.getInt(3));
         }
    } finally { if (conn != null) conn.close(); }
  }
  public static void main(String[] argv) { new ExempleJdbc(); }  }
```

JDBC - Java Database Connectivity

- Création de la base de données "*exemple*" dans MySQL.
  - Se connecter en "*root*" sur MySQL et créer la base de données "*exemple*".
  - Se connecter sur la base "*mysql*" et définir les droits d'un utilisateur :

```
C:/xampp/mysql/bin/mysql -u root -p
mysql> create database exemple;
mysql> use exemple;
mysql> show databases;
mysql> use mysql;
mysql> grant all privileges on exemple.*
       to 'login'@'localhost'
       identified by 'mot_de_passe_base_de_données'
       with grant option;
mysql> flush privileges;
mysql> exit;
```

  - Les données d'identifications (login et mot_de_passe) serviront dans une application Java pour la connexion du pilot JDBC à la BD.

JDBC - Java Database Connectivity

- Rappel des commandes SQL de base :
  - Créer une base de données : `CREATE DATABASE nom_BD;`
  - Activer une base de données : `USE nom_BD;`
  - Créer une table : `CREATE TABLE nom_table (liste d'attributs);`
  - Insérer des données dans une table : `INSERT INTO nom_table (liste d'attributs) VALUES (valeurs);`
  - Afficher le contenu d'une table : `SELECT * FROM nom_table;`
  - Afficher la structure d'une table : `DESCRIBE nom_table;`
- Description complète de SQL et ses commandes à l'adresse : `http://valk.iut-gtr.univ-mrs.fr/bdr`
- A consulter le site lié avec le cours est à l'adresse : `http://valk.iut-gtr.univ-mrs.fr/ic4`

2. JSP

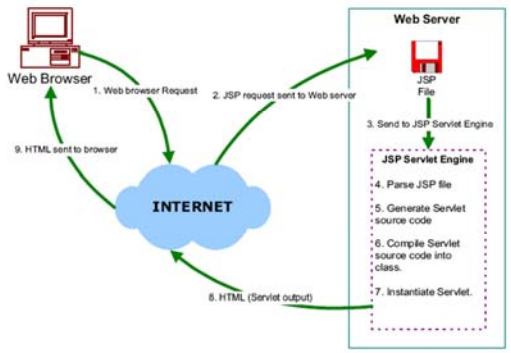
Java Server Pages

http://valk.iut-gtr.univ-mrs.fr/ic4

JSP - Introduction

- **Servlet et JSP** (*Java Server Pages*)
- *Servlet* est un programme Java s'exécutant côté serveur Web comme une classe "autonome" stockés dans un fichier *.class*.
- **JSP** est un source Java embarqué dans une page .html.
- **JSP** est un langage de script côté serveur.
- Une page JSP contient :
  - du contenu statique (texte simple, XHTML, XML, . . .)
  - du code JSP (Java) qui produit dynamiquement du contenu
- La page **JSP** est "exécutable" avec tous les serveurs Web auxquels on a ajouté un "moteur" de Servlet/JSP (ex: Tomcat)
- Une page **JSP** est compilée automatiquement en **Servlet** par le moteur de Servlets

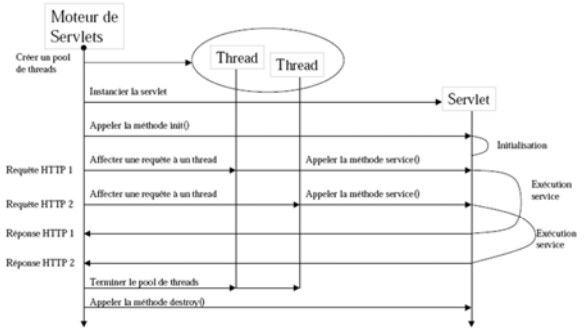
JSP - Architecture



JSP - Introduction

- En résumé :
  - **Servlet** : du code Java contenant XHTML
  - **JSP** : une page XHTML contenant du code Java
- Architectures des pages Web avec les JSP :
  - les parties statiques de la page Web sont écrites en XHTML
  - les parties dynamiques de la page Web sont écrites en Java
- Fonctionnement de JSP :
  - la page HTML est convertie en une Servlet
  - La Servlet est traitée par le moteur Java intégré au serveur Web (technologie des servlets) et retourne la page HTML construite au client

JSP - Fonctionnement



JSP - Introduction

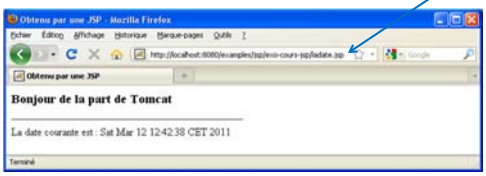
- Un exemple simple qui retourne la date du serveur Web :

```
<html>
<head>
<title>Obtenu par une JSP</title>
</head>
<body>
<h3>Bonjour de la part de Tomcat</h3>
<hr align="left" width="50%">
La date courante est : <%= new java.util.Date() %>
</body>
</html>
```

Instanciation de la classe Date en chemin complet

Appel du fichier "ladate.jsp" sur le port 8080 de Tomcat

Sauvegarde du contenu dans un fichier "ladate.jsp"



JSP - Eléments de syntaxe

- Les éléments suivants détiennent du code en Java
  - **<%@ ... %>** : directives valables pour la page. Exemple :

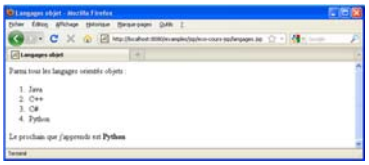
```
<%@ page contentType="text/plain; charset=UTF-8" %>
<%@ page import="java.io.*, java.util.*" %>
```

  - **<%! ... %>** : déclaration :
    Permet de définir des méthodes ou des données membres
  - **<% .... %>** : scriptlet (tests, itération, . . .) contient du code Java :
    insérer dans **_jspservice()** de la Servlet pour utiliser des objets prédéfinis :
    - **out :** le canal de sortie
    - **request (*HttpServletRequest*) :** l'objet requête
    - **response (*HttpServletResponse*) :** l'objet réponse
  - **<%= ... %>** : récupération d'une valeur d'expression
    l'expression Java renvoie un objet String ou un type primitif.
  - **<jsp: include ... />** : inclusion à l'exécution
  - **<jsp:forward ... />** : délégation à un autre composant

JSP - Eléments de syntaxe

- Un exemple :

```
<html>
<head>
<title>Langages objet</title>
</head>
<body>
<%!
  String[] langages =
  {"Java", "C++", "C#", "Python"};
  int random4() {
    return (int) (Math.random() * 4);
  }
%>
<p>Parmi tous les langages orientés objets :</p>
<ol>
<%
  for (int i=0; i < langages.length; i++) {
      out.println("<li>" + langages[i] + "</li>");
  }
%>
</ol>
<p>Le prochain que j'apprends est <b><%= langages[random4()] %> </b></p>
</body></html>
```



1. Partie déclarative
2. Partie Scriptlet
3. Partie expression

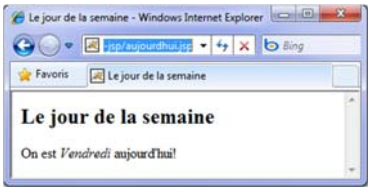
JSP - Un autre exemple

```
<html>
<head>
<title>Un exemple de page JSP</title>
</head>
<body>
<!-- définit les informations globales a la page
<%@page language="java" %>
<!-- Déclare la variable c -->
<%! char c = 0; %>
<!-- Scriplet (code java) %>
<%
for(int i = 0; i < 26; i++){
   for(int j = 0; j < 26; j++){
      c = (char)(0x41 + (26 - i + j)%26);
%>
<!-- Expression -->
<%= c %>
<% } %>
<br>
<% } %>
</body>
</html>
```

JSP - Un autre exemple

Le jour de la semaine

```
<%@ page
import="java.util.Date"
%>
<html>
<head>
<title>Le jour de la semaine
</title>
</head>
<body>
<h2>Le jour de la semaine</h2>
<%
// Les jours de la semaine
String jours[] = {"Dimanche", "Lundi", "Mardi", "Mercredi","Jeudi", "Vendredi",
"Samedi"};
// Objet de type Date
Date today = new Date();
// la valeur retournée (0 = Sunday, ..., 6 = Saturday)
int numero_jour = today.getDay();
out.println("<p>On est <i>" + jours[numero_jour] + "</i> aujourd'hui!");
%>
</body>
</html>
```



JSP - Paramètres d'exécution

- L'exécution d'une JSP peut être paramétrée par la directive :
  <%@ page buffer="none|xxxkb"%>
  pour spécifier la taille du buffer d'envoi de la réponse.
  Cela facilite le travail du serveur d'application et envoie la réponse plus rapidement au client.
- La directive : **<%@ page errorPage="file_name"%>**
  spécifie la page (JSP ou non) vers laquelle le serveur d'application renvoie lorsqu'une exception non gérée est lancée par la JSP.
- La directive : **<%@ page isErrorPage="true|false"%>**
  permet de spécifier une page d'erreur et autoriser ainsi la transmission de l'exception pour un éventuel traitement.
- Cela peut servir de mécanisme pour déboguer une page JSP !

JSP - Paramètres d'exécution

- Enchaîner les pages :
  Un page JSP peut en appeler une autre par la directive :
  <jsp:forward>
- Syntaxe :
  <jsp:forward page="pageDeRedirection" />
- Exemple :

```
<% String repUtilisateur = request.getParameter("repTextField");
   int rep = Integer.parseInt(repUtilisateur);
   if ((rep % 2) == 0) {
%>
     <jsp:forward page="positif.jsp"/>
<% } else { %>
     <jsp:forward page="negatif.jsp"/>
<% } %>
```


JSP - Directives

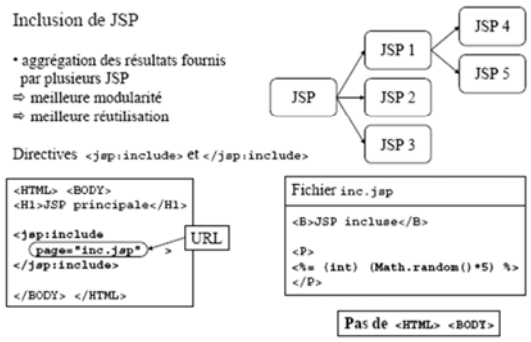
- **<%@ page ... %>** Donne des informations sur la JSP (non obligatoire, valeurs par défaut)
- **<%@ page import="..."%>** (ex. <%@ *page import="java.io.*"*%>) les "import" nécessaires au code Java de la JSP
- **<%@ page errorPage="..."%>** fournit l'URL de la JSP à charger en cas d'erreur
- **<%@ page contentType="..."%>** le type MIME du contenu retourné par la JSP
  (ex. <%@ *page contentType="text/html"*%>)
- **<%@ page isThreadSafe="..." %>** *true* ou *false* : la JSP peut être exécutée par plusieurs clients à la fois (valeur *true* par défaut)
- **<%@ page isErrorPage="..." %>** *true* ou *false* : la JSP est une page invoquée en cas d'erreur (*true*)

JSP - Objets implicites

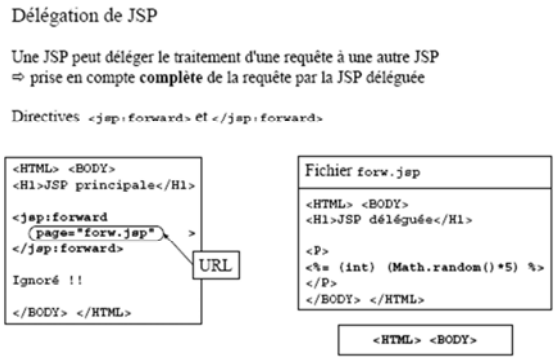
Objets pré-déclarés utilisables dans le code Java des JSPs :

- **out** : le flux de sortie pour générer le code HTML
- **request** : la requête qui a provoqué le chargement de la JSP
- **response** : la réponse à la requête de chargement de la JSP
- **page** : l'instance de servlet associée à la JSP courante (= this)
- **exception** : l'exception générée en cas d'erreur sur une page
- **session** : suivi de session pour un même client
- **application** : espace de données partagé entre toutes les JSP

JSP - Inclusion



JSP - Délégation



JSP - Délégation et inclusion

Délégation et inclusion de JSP

Récupération des paramètres
≡ à la récupération des paramètres transmis via un formulaire

```
<HTML> <BODY>
<H1>JSP déléguée/incluse</H1>

Nom : <%= request.getParameter("nom") %>
Prénom : <%= request.getParameter("prenom") %>

</BODY> </HTML>
```


JSP - Compléments

Compléments sur l'API

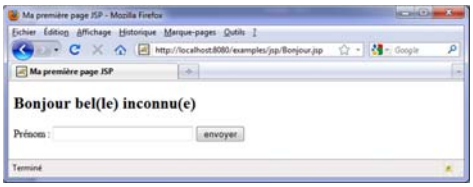
Méthodes appelables sur l'objet prédéfini `request`

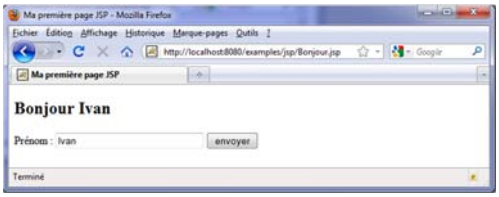
- `String getProtocol()`
  retourne le protocole implanté par le serveur (ex. : HTTP/1.1)
- `String getServerName() / String getServerPort()`
  retourne le nom/port de la machine serveur
- `String getRemoteAddr() / String getRemoteHost()`
  retourne l'adresse/nom de la machine cliente (ayant invoqué la servlet)
- `String getScheme()`
  retourne le protocole utilisé (ex. : http ou https) par le client

JSP - Un exemple complet

```
<!-- Fichier Bonjour.jsp -->
<%@ page contentType="text/html; charset=ISO-8859-1" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html><head><title>Ma première page JSP</title></head>
<body>
<!-- Traitement du formulaire -->
<% String prenom=request.getParameter("prenom"); %>
<h2>Bonjour <%=(prenom!=null && prenom.length()!=0)?prenom:"bel(le)
inconnu(e)"%></h2>
<% if (prenom!=null && prenom.equals("le monde")) { %>
<h2>Bonjour à toi</h2>
<% } %>
<!-- Formulaire à remplir -->
<form action="Bonjour.jsp" method="post">
<label>Prénom : </label><input type="text" name="prenom" size="30">
<input type="submit" value="envoyer">
</form>
</body>
</html>
```


JSP - Un exemple complet





JSP - Traitement de formulaire

- Les pages JSP peuvent utiliser les fonctionnalités de la classe *javax.servlet* (puisqu'elles sont traduites en un servlet).
- On dispose de certains objets sous forme de variables :
  - *request* de la classe *javax.servlet.http.HttpServletRequest*
  - *out* de la classe *javax.servlet.ServletOutputStream*
  - *response* de la classe *javax.servlet.http.HttpServletResponse*
- Pour récupérer une variable "formulaire" utiliser:
  - `request.getParameter("paramètre")`
    - *paramètre* est le nom de l'attribut qui vient des requêtes GET, POST, DELETE et PUT.
    - *request* possède d'autres méthodes utiles.

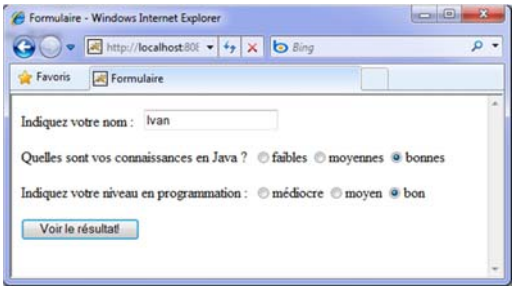
JSP - Traitement de formulaire

- Exemple de formulaire (Partie XHTML, fichier "*formulaire.html*")

```
<html><head><title>Formulaire</title></head><body>
<form action="sondage.jsp" method=post>
<p>Indiquez votre nom :  
<INPUT TYPE="text" NAME="nom" MAXLENGTH="50">
</p>
<p>Quelles sont vos connaissances en Java ? 
<input type="radio" name="choice1" value="1" checked>faibles
<input type="radio" name="choice1" value="2">moyennes
<input type="radio" name="choice1" value="3">bonnes
</p>
<p>Indiquez votre niveau en programmation : 
<input type="radio" name="choice2" value="1">médiocre
<input type="radio" name="choice2" value="2" checked>moyen
<input type="radio" name="choice2" value="3">bon
</p>
<P>
<input type="submit" value="Voir le résultat!"></p>
</form>
</body></html>
```

JSP - Traitement de formulaire

- Exemple de formulaire



JSP - Traitement de formulaire

- Exemple de formulaire (Partie JSP, fichier "*sondage.jsp*"))

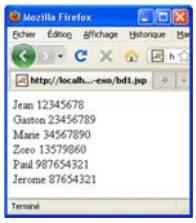
```
<%
// recupère chaines de caractères
String choice1 = request.getParameter("choice1");
String choice2 = request.getParameter("choice2");
String nom = request.getParameter("nom");
out.println(nom + ", votre input était: question a = " + choice1 +
" ,question b = " + choice2);
// Convertir les choix en entiers
int score = Integer.parseInt(choice1) + Integer.parseInt(choice2);
out.println("<h3>Votre score est de " + score + "</h3>");
if (score < 3) {
  out.print ("<p>Vous êtes un débutant</p>");
} else if (score < 5) {
  out.print ("<p>Vous avez un niveau moyen</p>");
} else {
  out.print ("<p>Vous êtes un expert !</p>");
}
%>
```


JSP - Traitement de formulaire

- Exemple de formulaire

JSP et JDBC

- Appel de JDBC par JSP :




```
<%@ page import="java.sql.*" %>
<%
String connectionURL =
       "jdbc:mysql://localhost:3306/hotel";
Connection connection = null;
Statement statement = null;
ResultSet rs = null;
%>
<html>
<body>
<%
Class.forName("com.mysql.jdbc.Driver").newInstance();
connection = DriverManager.getConnection(connectionURL,"user","123456");
statement = connection.createStatement();
rs = statement.executeQuery("SELECT * FROM clients");
while (rs.next()) {
    out.println(rs.getString("nom")+" "+rs.getInt(4)+"<br>");
}
rs.close();
%>
</body>
</html>
```


3. Java 2 Micro Edition (J2ME)

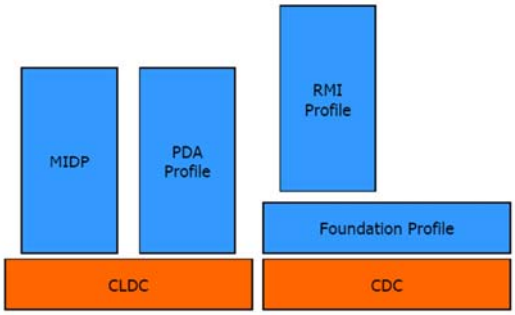
Développement d'applications mobiles

J2ME - Java 2 Micro Edition

- J2ME définit deux grandes familles d'appareils :
  - *Appareils à fonctionnalités dédiées ou limitées* : ressources et interface graphique limitées, peuvent se connecter par intermittence au réseau *CLDC* (*Connected Limited Device Configuration* : JSR30 - mémoire 128KB à 512KB).
    (exemple : téléphone mobile, agenda électronique, PDA, pagers, ...)
  - Appareils proposant une interface graphique riche, possédant une connexion continue au réseau *CDC* (*Connected Device Configuration* : JSR36 - RAM > 512 Kb)
    (exemple : PDA haut de gamme, Smartphone, système de navigation, ...)
- J2ME est modulaire grâce à trois types d'entités :
  - *Les configurations* : définissent des caractéristiques minimales d'un large sous type de matériel, d'une machine virtuelle et d'API de base
  - *Les profiles* : définissent des API relatives à une fonctionnalité commune à un ensemble d'appareils (exemple : interface graphique, ...)
  - *Les packages optionnels* : définissent des API relatives à une fonctionnalité spécifique dont le support est facultatif

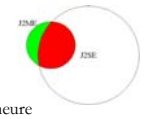
J2ME - Java 2 Micro Edition

- Configuration et Profile



J2ME - Java 2 Micro Edition (CLDC)

- De nombreuses classes sont définies dans J2SE et J2ME mais souvent elles possèdent moins de fonctionnalités dans l'édition mobile.
- L'API du CLDC se compose de quatre packages :



  - *java.io* : classes pour la gestion des entrées - sorties par flux
  - *java.lang* : classes de base du langage java
  - *java.util* : classes utilitaires pour gérer les collections, la date et l'heure
  - *javax.microedition.io* : classes pour gérer des connections génériques
- L'API du MIDP se compose des API du CDLC et de trois packages
  - *javax.microedition.midlet* : cycle de vie de l'application
  - *javax.microedition.lcdui* : interface homme machine
  - *javax.microedition.rms* : persistance des données
- Les applications créées avec MIDP sont des *midlets* :
  - classes qui permettent le dialogue entre le système et l'application.

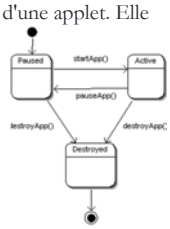
J2ME - Java 2 Micro Edition (MIDlet)

- Une *midlet* possède trois méthodes gérant le cycle de vie de l'application en trois états possibles (active, suspendue ou détruite) :
  - *startApp()* : méthode appelée à chaque démarrage ou redémarrage de l'application
  - *pauseApp()* : cette méthode est appelée lors de la mise en pause de l'application
  - *destroyApp()* : cette méthode est appelée lors de la destruction de l'application
- Les trois méthodes doivent obligatoirement être redéfinies.

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class Test extends MIDlet {
   public Test() {  //Constructeur
   }
   public void startApp() { // début d'exécution
   }
   public void pauseApp() { // en pause
   }
   public void destroyApp(boolean unconditional) {
   }
}
```


J2ME - Java 2 Micro Edition (MIDlet)

- Le cycle de vie d'une MIDlet est semblable à celui d'une applet. Elle possède plusieurs états :
  - *paused :*
  - *active :*
  - *destroyed :*



- Le changement de l'état de la MIDlet peut être provoqué par l'environnement d'exécution.
- La méthode *startApp()* est appelée lors du démarrage ou redémarrage de la MIDlet.
- La méthode *pauseApp()* est appelée lors de mise en pause de la MIDlet.
- La méthode *destroyApp()* est appelée juste avant la destruction de la MIDlet.

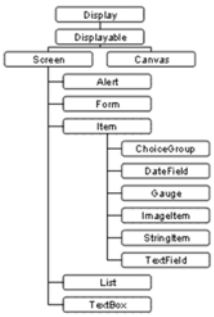
J2ME - MIDlet - Interface Utilisateur

- L'interface utilisateur (IHM ou GUI)
  - Les possibilités des *midlets* sont très réduites pour permettre une exécution sur un maximum de machines allant du téléphone portable au PDA.
- La classe *Display*
  - Pour utiliser les éléments graphiques, il faut instancier un objet de type *Screen*.
  - C'est un objet du type *Display* qui possède des méthodes pour afficher les éléments graphiques.
  - La méthode statique *getDisplay()* renvoie une instance de la classe *Display* qui encapsule l'écran associé à la *midlet* fournie en paramètre de la méthode.

```
public class Bonjour extends MIDlet {
   private Display display;
   public Bonjour() {
      display = Display.getDisplay(this);
   }
}
```


J2ME - MIDlet - Interface Graphique

- Les éléments de l'IG appartiennent à une hiérarchie d'objets :
  - Tous les éléments affichables héritent de la classe abstraite *Displayable*.
  - La classe *Screen* est la classe mère des éléments graphiques de haut niveau.
  - La classe *Canvas* est la classe mère des éléments graphiques de bas niveau.
  - Tout élément graphique mis dans un *Display* hérite de *Displayable*.
  - Un seul objet de type *Displayable* peut être affiché à la fois.
  - La classe *Display* possède la méthode *getCurrent()* pour connaître l'objet courant affiché et la méthode *setCurrent()* pour afficher l'objet fourni en paramètre.



J2ME - MIDlet - la classe TextBox

- La classe *TextBox*
  - *TextBox* est un composant de type *Screen* et permet l'affichage et la modification d'un texte à l'écran.

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class MIDletBonjour extends MIDlet {

   private Display display;
   private TextBox textbox;

   public MIDletBonjour() {
      this.display = Display.getDisplay(this);
      this.textbox = new TextBox("", "Bonjour", 20, 0);
   }
   public void startApp() {
      display.setCurrent(textbox);
   }
   public void pauseApp() {
   }
   public void destroyApp(boolean unconditional) {
   }
}
```


J2ME - MIDlet - La classe List

- La classe *List*
  - La classe permet la sélection d'un ou plusieurs éléments dans une liste d'éléments.

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
public class MIDletList extends MIDlet {
   private Display display;
   private List liste;
   protected static final String[] items =
            {"Item 1","Item 2","Item 3","Item 4"};
   public MIDletList() {
      this.display = Display.getDisplay(this);
      this.liste = new List("Selection", List.EXCLUSIVE,
                            items, null);
   }
   public void startApp() {
      display.setCurrent(liste);
   }
   public void pauseApp() {
   }
   public void destroyApp(boolean unconditional) {
   }
}
```

J2ME - MIDlet - La classe Forme

- La classe *Form*
  - La classe *From* sert de conteneur et permet d'insérer dans l'élément graphique qu'elle représente d'autres éléments graphiques de type *Item*.
- La classe *Item*
  - La classe *Item* est la classe mère de tous les composants graphiques qui peuvent être insérés dans un objet de type *Form*. La classe définit deux méthodes : *getLabel()* et *setLabel()*.
- Composants qui héritent de la classe *Item*
  - *ChoiceGroup* : sélection d'un ou plusieurs éléments
  - *DateField* : affichage et saisie d'une date
  - *Gauge* : affichage d'une barre de progression
  - *ImageItem* : affichage d'une image
  - *StringItem* : affichage d'un texte
  - *TextField* : saisie d'un texte

J2ME - MIDlet - La classe Forme

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
public class MidletFormItem extends MIDlet {
    private Display display;
    private Form form;
    private ChoiceGroup choiceGroup;
    private DateField dateField;
    private DateField timeField;
    private Gauge gauge;
    private StringItem stringItem;
    private TextField textField;
    public MidletFormItem() {
        this.display = Display.getDisplay(this);
        this.form = new Form("Ma form");
        String choix[] = {"Choix 1", "Choix 2"};
        stringItem = new StringItem(null,"Mon texte");
        choiceGroup = new ChoiceGroup("Sélectionner",
                              Choice.EXCLUSIVE,choix,null);
        dateField = new DateField("Heure",DateField.TIME);
        timeField = new DateField("Date",DateField.DATE);
        gauge = new Gauge("Avancement",true,10,1);
        textField = new TextField("Nom","Votre nom",20,0);
        form.append(stringItem); form.append(choiceGroup);
        form.append(timeField);  form.append(dateField);
        form.append(gauge);      form.append(textField);  }
    public void startApp() { display.setCurrent(form); }
    public void pauseApp() {    }
    public void destroyApp(boolean unconditional) { }
}
```

J2ME - MIDlet - Les événements

- La gestion des événements
  - Un objet de la classe *Command* est un "bouton MIDP" que l'utilisateur va pouvoir actionner à l'aide des touches clavier.
  - Les *Displayable* : *Screen*, *TextBox*, etc. possèdent une méthode : *public void addCommand(Command);*
  - Le bouton va être ajouté dans l'interface graphique du *Displayable* en fonction de nombre de boutons, type de l'écran ou de téléphone mobile.
  - La classe *Command* possède un seul constructeur : *public Command(String label, int type, int priority);*
    - *label* : le texte du bouton;
    - *type* : est une constante de la classe *Command*.
      - *OK* : suggère le lancement d'un traitement; *BACK* : doit ramener à l'écran précédent
      - *CANCEL* : suggère de ne pas lancer un traitement; *STOP* : suggère d'arrêter un traitement
      - *EXIT* : doit arrêter la MIDlet; *HELP* : doit afficher une aide.
    - *priority* : les petites valeurs amènent une *Command* mieux placée dans l'interface.

J2ME - MIDlet - Les événements

- La programmation des traitements des événements est similaire à J2SE
  - On associe un (seul) *listener* au composant.
  - Le *listener* lance une méthode convenue lorsque la *Command* associé au *Displayable* a été manipulée par l'utilisateur.
  - L'association est faite par : *public void setCommandListener(CommandListener l);*
  - La méthode lancée par le *listener* est : *public void commandAction(Command c, Displayable d);*
    - Le premier argument indique la *Command* de l'interface graphique qui a été utilisée
    - Une *Command* peut être associée à plusieurs *Displayable*, le second argument indique le *Displayable* qui contient la *Command* actionnée par l'utilisateur
  - La méthode *setCommandListener()* est lancée sur le *Displayable* contenant la *Command*.

J2ME - MIDlet - Les événements

```
import java.util.*;
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
public class dateMidlet extends MIDlet implements CommandListener {
    private Command exitCommand; // The exit command
    private Display display;     // The display for this MIDlet
    public dateMidlet() {
        display = Display.getDisplay(this);
        exitCommand = new Command("Exit", Command.EXIT, 0);
    }
    public void startApp() {
        String str = null;
        Date date = new Date();
        str = "Date et Heure : " + date.toString();
        TextBox t = new TextBox("Date", str, 256, 0);
        t.addCommand(exitCommand);
        t.setCommandListener(this);
        display.setCurrent(t);
    }
    public void pauseApp() {   }
    public void destroyApp(boolean unconditional) {    }

    public void commandAction(Command c, Displayable s) {
        if (c == exitCommand) {
            destroyApp(false);
            notifyDestroyed();
        }
    }
}
```

J2ME - MIDlet - Un exemple complet

```
import javax.microedition.lcdui.*;
import javax.microedition.midlet.MIDlet;
public class TextBoxInteractionMIDlet extends MIDlet implements CommandListener {
    private static final int MAX_TEXT_SIZE = 64;
    protected TextBox textBox;     protected Display display;
    protected boolean started;     Command exitCommand, reverseCommand;
    protected void startApp() {
        if (!started) {
        String str = null;          str = "Programme avec TextBox";
        textBox = new TextBox("TextBox Example", str, MAX_TEXT_SIZE,
                                    TextField.ANY);
        exitCommand = new Command("Exit", Command.EXIT, 0);
        reverseCommand = new Command("Reverse", Command.OK, 0);
        textBox.addCommand(exitCommand);
        textBox.addCommand(reverseCommand);
        textBox.setCommandListener(this);
        display = Display.getDisplay(this);
        display.setCurrent(textBox);
        started = true;     }     }
    protected void pauseApp() {}
    protected void destroyApp(boolean unconditional) { }
    public void commandAction(Command cmd, Displayable d) {
        if (cmd == exitCommand) {
          destroyApp(true);
          notifyDestroyed();
        } else if (cmd == reverseCommand) {
          String text = textBox.getString();
        if (text != null) {
          StringBuffer str = new StringBuffer(text);
          textBox.setString(str.reverse().toString());
        }
        }    }    }
```

MIDlet - installation sur téléphone mobile

- Une application J2ME (MIDlet) après compilation est constituée de deux fichiers : *application.jar* et *application.jad*.
  - On peut transférer les fichiers de l'application j2me sur le téléphone mobile du PC par voie GSM, Internet, câble, infrarouge ou Bluetooth afin de les installer.
- Il y a, en générale, 4 modes de déploiement d'une application j2me :
  - *Over-the-Air* (OTA) ou à partir d'un Serveur Web :
    - Les fichiers à installer sont accessibles sur un serveur Web qui reconnait les types MIME : *.jar: application/java-archive; .jad: text/vnd.sun.j2me.app-descriptor,* alors les fichiers sont téléchargés par le navigateur mobile du téléphone. L'installation commence au lancement du côté unité mobile du fichier *.jad.*
  - *IR* ou *Bluetooth* peuvent être utilisés pour envoyer la MIDlet du PC à une unité mobile. L'installation alors se déroule de la même manière (voir OTA).
  - L'envoi de la MIDlet par câble USB nécessite un logical de communication avec l'unité mobile ("*Ovi suite*" ou "*PC suite*" pour la marque Nokia).
  - Un "*WAP Push messages*" sous la forme de hyperlien pour télécharger la MIDlet.

MIDlet - Les classes Image et ImageItem

- Deux classes pour manipuler des images :
  - La classe *Image* : Crée un objet image et rapporte les attributs de l'image tels que : hauteur et largeur
  - La classe *ImageItem* : Affiche l'image dans un conteneur de type Forme et rapporte les attributs d'alignement de l'image
- Une MIDlet autorise deux types d'image :
  - *fixe* : qui ne peut pas être modifiée après création
  - *mutable* : image créée et stockée en mémoire
- Afficher une image :
  - *Image.createImage(String name);*
  - *ImageItem(String label, Image img, int layout, String altText);*
  - Pour afficher l'image il faut ajouter le constructeur dans un conteneur de type Forme.

J2ME - MIDlet - Afficher une image

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
public class imageMidlet extends MIDlet implements CommandListener {
  private Command cmdExit;
  private Display display = null;
  private Form mainForme;
  private StringItem Msg;
  private Ticker t;
  public imageMidlet() {
    display = Display.getDisplay(this);
    cmdExit = new Command("Exit", Command.EXIT, 1);
    mainForme = new Form("ma Forme");
    t = new Ticker("Le Rafale");
    Msg = new StringItem("", " - Le Rafale - ");
    mainForme.addCommand(cmdExit);
    mainForme.append(Msg);
    mainForme.setCommandListener(this);
    try {
        Image img = Image.createImage(
              (display.isColor()) ? "/rafale.png" : "/img.png");
      mainForme.append(new ImageItem(null, img,
            ImageItem.LAYOUT_CENTER, null));
      display.setCurrent(mainForme);
    } catch (java.io.IOException e) { e.printStackTrace(); } }
  public void startApp() {
    mainForme.setTicker(t);
    display.setCurrent(mainForme);    }
  public void pauseApp() {    }
  public void destroyApp(boolean unconditional) {    }
  public void commandAction(Command c, Displayable s) {
    if (c == cmdExit) { destroyApp(false); notifyDestroyed();
    }   }}
```

Déposer l'image dans les répertoires 'src' et 'dist' du projet j2me

MIDlet - Les classes Graphics et Canvas

- La classe *Graphics* propose les méthodes de dessin de :
  - Rectangles, Ellipses, Lignes, Textes, Images, …
- La classe *Canvas* propose :
  - La surface de dessin et les méthodes "call-back" d'interactivité
- Le dessin peut se faire
  - Soit directement à l'écran
  - Soit dans une image : En appelant la méthode *getGraphics()* de la classe *Image*
- La couleur est codée sur 3 octets : **`0x00RRGGBB`**
  - **`0x00FF0000`** : Rouge
  - **`0x0000FF00`** : Vert
  - **`0x000000FF`** : Bleu

J2ME - MIDlet - Afficher un dessin

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
public class dessinMidlet extends MIDlet implements CommandListener {
  private Canvas feuille;
  public dessinMidlet() {
    feuille = new Canvas() {
      public void paint(Graphics g) {
        g.setColor(0xFFFFFF);
        g.fillRect(0,0,getWidth(),getHeight());
        g.setColor(0xFF0000);
        g.drawLine(5,5,60,60);
        g.setColor(0x00FF00);
        g.drawRect(30,30,100,50);
        g.setColor(0x0000FF);
        g.fillArc(60,60,50,50,0,360);
        g.setColor(0xFF00FF);
        g.drawString("Figures",100,10, Graphics.TOP | Graphics.LEFT);
      }
    };
  }
  public void startApp() {
    Display.getDisplay(this).setCurrent(feuille);
  }
  public void pauseApp() {}
  public void destroyApp(boolean unconditional) {}
  public void commandAction(Command c, Displayable s) {
    if (c.getCommandType() == Command.EXIT)
      notifyDestroyed();
  }
}
```

MIDlet - La classe Canvas

- On peut interagir avec le *Canvas* :
  - Avec les touches du clavier;
  - Avec un dispositif de pointage;
  - Avec des touches de jeu.
- Pour avoir un retour sur les actions des touches, il faut surcharger les méthodes dérivées de la classe *Canvas* :
  - Pour être prévenu qu'une touche est pressée :
    - La méthode *keyPressed(int keyCode)*
  - Pour être prévenu qu'une touche est maintenue enfoncée :
    - La méthode *keyRepeated(int keyCode)*
  - Pour être prévenu qu'une touche est relâchée :
    - La méthode *keyReleased(int keyCode)*

MIDlet - La classe Canvas

- Le *Canvas* fournit des constantes pour nommer les touches :
  - Pour les chiffres :
    KEY_NUM0, …, KEY_NUM9
  - Pour le # :
    KEY_POUND
  - Pour les touches de direction :
    LEFT, RIGHT, UP, DOWN
  - Pour les touches de jeu :
    GAME_A, GAME_B, GAME_C, GAME_D, FIRE

J2ME - MIDlet - Un petit jeu

```
import javax.microedition.lcdui.*;
import javax.microedition.midlet.*;
public class jeuxMidlet extends MIDlet implements CommandListener {
  private Canvas jeu;   private Display dsp;   private Command cmdq;
  public jeuxMidlet() {
    cmdq = new Command("Exit", Command.EXIT, 0);
    jeu = new Canvas() {       // Définir le Canvas et les coordonnées de début
        int x = getWidth()/2;    int y = 150;  // du jeu de raquette
        public void paint(Graphics g) {
          g.setColor(0xFFFFFF);    // Définir la zone du jeu
          g.fillRect(0,0,getWidth(),getHeight());
          g.setColor(0xFF0000);
          g.fillRect(x,y,20,5);    // Définir la raquette comme un rectangle
        }
        public void keyPressed(int key) {  // Traitement des touches 1 et 3,
          if (key == Canvas.KEY_NUM1) {   // rafraichir l'écran pour
            if (x > 5) { x-=5;   repaint();  }  // obtenir la position
          } else if(key == Canvas.KEY_NUM3) {    // suivante de la raquette
            if (x < getWidth()-5) {  x+=5;  repaint();  }
          }       }
    };
  }
  public void commandAction(Command c, Displayable s) {
    if (c == cmdq) { notifyDestroyed(); }
  }
  public void startApp() {
    dsp = Display.getDisplay(this);
    dsp.setCurrent(jeu);  }
  public void pauseApp() { }
  public void destroyApp(boolean b) { }
}
```

4. Android

Développement d'activités

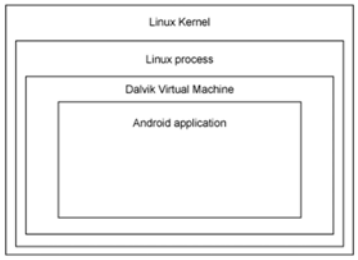
Java sous Android

Fonctionnement du système Android

- Android est un système d'exploitation basé sur Linux et créé par l'*Open Handset Alliance* dirigée par *Google*.
- Android propose une interface de programmation Java avec sa propre machine virtuelle DVM (*Virtual Machine Dalvik*). L'interface fournit des outils pour la compilation, le débogage et un simulateur de périphérique mobile et embarqué.
- Android utilise une machine virtuelle spéciale, ce qui rend son *bytecode* incompatible avec celui de Java standard. Par conséquent, un outil "dx" est proposé pour convertir un fichier Java classe dans le format Android "dex" (*Dalvik exécutable*).
- Une applications Android est emballée dans un fichier. apk (*Android Package*) par AAPT (*Android Asset Packaging Tool*) Pour développer Google fournit ADT (*Android Development Tools*) pour Eclipse et pour NetBeans de Sun (Oracle).

Le système Android

- Architecture d'une application Java Android



Fonctionnement du système Android

- L'ADT effectue automatiquement la conversion d'une classe "*dex*" en *.apk* au cours du déploiement.
- Android supporte le graphisme 2-D et 3-D en utilisant la bibliothèque *OpenGL*.
- Le stockage de données dans une BD est pris en charge par *SQLite*.
- *SQLite* est une *Open Source Database* intégrée dans Android.
  - SQLight supporte les fonctionnalités standards pour une BDR telles que SQL syntaxe, la gestion des transactions et "*prepared statements*".
- Une application sous Android s'exécute dans son propre processus et sous son propre nom d'utilisateur qui est généré automatiquement au cours de son déploiement. Par conséquent, l'application est isolée des autres applications en cours et ne peut pas facilement affecter les autres applications.

Une application Android

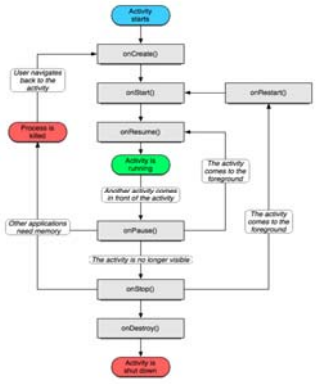
- Activity : La couche représentative et visuelle de l'application qui peut avoir plusieurs couches qui alternent entre elles lors de l'exécution.
- Views : Le IHM (GUI) est une "*widgets*" couche qui hérite des classes "*android.view.View*" et "*android.view.ViewGroups*".
- Service : Il n'a pas de vue mais permet l'exécution d'un algorithme sur un temps indéfini et terminé en fonction de la tâche.
- Content Provider : fournit des données aux applications, via un fournisseur de contenu capable de les partager avec d'autres applications (agenda, photos, contacts stockés dans le téléphone).
- Intents : Une application peut appeler un service ou une activité (explicite) ou appeler un service du système Android (implicites).
- Broadcast Receiver : reçoit les messages système et les "Intents" implicites.

Une application Android

- Android étant un système d'exploitation pour téléphone portable de nouvelle génération dispose d'un kit de développement (SDK) basé sur le langage Java.
- Le SDK est disponible pour les plateformes Linux, Mac et Windows à l'adresse : *http://code.google.com/android/download.html*
- Pour développer avec l'IDE Eclipse Google fournit un plugin ADT (*Android Development Tools*).
- Pour le développementt avec l'IDE NetBeans Android propose le plugin "*nbandroid*" accessible à : *http://nbandroid.kenai.com*.
- Le développement pour Android est possible aussi sans un IDE particulier en se servant des commandes du SDK d'Android avec Ant pour la compilation et la gestion du simulateur.
- A consulter : *http://ydisanto.developpez.com/tutoriels/android/debuter/*

Cycle de vie d'une application Android

```
public class Activity extends AppContext {
    protected void onCreate(Bundle savedIState);
    protected void onStart();
    protected void onRestart();
    protected void onResume();
    protected void onPause();
    protected void onStop();
    protected void onDestroy();
}
```



Architecture d'une application Android

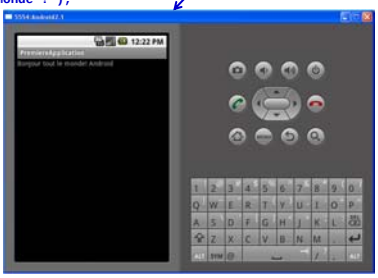
- onCreate : La méthode est appelée à la création d'une activité pour initialiser les données nécessaires à l'exécution de l'application. A l'appel de la méthode un *Bundle* est passé en argument. Ce Bundle contient l'état de sauvegarde enregistré lors de la dernière exécution.
- onStart : La méthode est appelée dans le cas où l'application est en arrière-plan et qu'elle repasse en avant-plan. Si l'activité ne peut pas passer en avant plan alors, l'activité sera transférée à OnStop.
- onResume : La méthode est appelée après OnStart quand l'application passe en background à cause d'une autre application.
- onPause : La méthode met en pause l'application et se relance avec la méthodes OnResume.
- onStop : Appelée quand l'activité n'est plus visible.
- onDestroy : Appelée quand l'application est fermée (*processus closed*).

Android - Java : Bonjour

```
import android.app.Activity;
import android.os.Bundle;
import android.widget.TextView;
public class Bonjour extends Activity {
  public void onCreate(Bundle icicle) {
    super.onCreate(icicle);
    TextView tv = new TextView(this);
    tv.setText("Bonjour tout le monde !");
    setContentView(tv);
  }
}
```

Simulateur d'unité mobile sous Android

Une première application "Hello World"



Programmer sous Android

- Interface graphique par programmation
  - Pour faciliter le développement Android propose un grand nombre de "*widgets*" qui sont des éléments d'interface graphique qu'on peut utiliser dans une application de manière directe et simple.
  - On peut utiliser les classiques :
    - boutons, listes déroulantes, cases à cocher
  - mais aussi de composants plus poussés :
    - des horloges, des sélecteurs de date, des galerie photos et des afficheurs de vidéo.
- Interface graphique par fichier XML
  - Le fichier XML sera lu par le programme et l'interface graphique sera automatiquement générée en conséquence. Il devient ainsi beaucoup plus facile de modifier et de faire évoluer une interface graphique déjà existante.

Android - Un deuxième exemple Java

```
package org.me.androidapplication2;
import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import java.util.Date;
public class Now extends Activity implements View.OnClickListener {
   Button btn;
   @Override
   public void onCreate(Bundle icicle) {
         super.onCreate(icicle);
         btn = new Button(this);
         btn.setOnClickListener(this);
         updateTime();
         setContentView(btn);
   }
   public void onClick(View view) {
         updateTime();
   }
   private void updateTime() {
         btn.setText(new Date().toString());
   }
}
```

Déroulement de l'exemple (1)

- La déclaration de paquetage doit être identique à celle utilisée pour créer le projet.
- Pour un projet Java il faut importer les classes auxquelles l'application fait référence.
  - La plupart des classes spécifiques à Android se trouvent dans le paquetage *android*
  - Les classes de Java SE sont utilisables par les programmes Android, mail il faut consulter le guide de référence des classes Android pour connaitre leur disponibilité.
- Les activités sont des classes publiques héritées de la classe de base *android.app.Activity*.
- Les *widgets* sont des éléments d'interface graphique qu'on peut utiliser dans une application.

Déroulement de l'exemple (2)

- L'activité contient un bouton : **`Button btn;`**
  - Un bouton est un *widget* Android et peut être utilisé dans une application.
- Pour capturer tous les clics de bouton dans l'activité elle-même on implémente **`OnClickListener`**.
- La méthode **`onCreate()`** est appelée au lancement de l'activité, alors on établi un chaînage vers la *superclasse* afin d'initialiser l'activité Android de base.
- L'instance de bouton créée (**`new Button(this)`**), on demande l'envoie de tous les clics sur ce bouton à l'instance de l'activité (**`setOnClickListener()`**).
- Un appel la méthode privée **`updateTime()`** est constitué, et pour finir on configure la vue du contenu de l'activité avec le bouton lui-même (**`setContentView()`**).

Déroulement de l'exemple (3)

- Tous les *widgets* dérivent de la classe de base *View*.
- **`Bundle icicle`** est un gestionnaire opaque, que toutes les activités reçoivent lors de leur création.
- Avec *Swing*, un clic sur un *JButton* déclenche un *ActionEvent* qui est transmis à l'*ActionListener* configuré pour ce bouton.
- Avec Android un clic sur un bouton fait appel de la méthode **`onClick()`** sur l'instance **`OnClickListener`** configurée pour ça.
- L'écouteur reçoit la vue qui a déclenché le clic et on fait alors appel à la méthode privée **`updateTime()`**.
- L'ouverture de l'activité (*onCreate()*) ou un clic sur le bouton (*onClick()*) doit provoquer la mise à jour du label du bouton avec la date courante. On utilise pour cela la méthode **`setText()`**, qui fonctionne exactement comme avec les *JButton* de *Swing*.

Layout XML (1)

- *Android,* vis-à-vis des autres systèmes d'exploitation mobiles, possède la possibilité de créer des *interfaces graphiques* à l'aide de *fichiers XML*.
- Cette particularité favorise la séparation de la description de l'interface graphique (*layout XML*) de la logique applicative (*code Java*).
- Cela a pour effet la séparation du *fond* de la *forme* et facilite par exemple : la "localisation" d'une interface graphique en fonction de la langue (*français, anglais, bulgare*), du contexte d'utilisation (*jour ou nuit*) ou la modification de l'ergonomie (*boutons, listes, cases à cocher*).
- *Android* inclut un système proche des CSS bien connu pour le développement Web. Il s'agit des *styles* et des *thèmes* qui permettent le respect d'une cohérence à travers une application.
- Ainsi, l'interface graphique est construite dans des fichiers XML présents dans le dossier *res/layout* d'un projet Android (*Eclipse*).

Layout XML (2)

- La structure générale des *XML layout* d'Android est une arborescence d'éléments où chaque nœud porte le nom d'une classe de type *View*.
- Cette structure permet de créer rapidement des UI et de les maintenir facilement par la suite.

Attribut	Explication
xmlns:android	Cet attribut doit obligatoirement être présent au début de tout fichier de layout Android. C'est la déclaration XML namespace qui demande aux outils Android de se référer aux attributs définis.
android:id	Cet attribut assigne un identifiant unique à l'élément TextView
android:layout_width	Cet attribut définit la largeur de la zone d'affichage en se référent à l'espace écran disponible. Dans notre cas, nous allons utiliser toute la largeur de l'écran
android:layout_height	Tout comme le android:layout_width, sauf que ça concerne la hauteur
android:text	Ca permet de définir le texte à afficher par TextView. Dans cet exemple, nous utilisons une chaîne de caractère de type "ressource" au lieu d'une chaîne "Hard-coded". La valeur de la chaîne de caractère "hello", est définie dans le fichier res/values/strings.xml. Il est recommandé d'utiliser cette technique en vue de faciliter la traduction des applications Android

Android : Styles et Thèmes

- Un *style* est un ensemble d'attributs de formatage qu'on peut appliquer à des éléments simples mis dans un fichier XML.
  - Par exemple, on peut définir un style qui spécifie une *taille* ou une *couleur* appliqué à un certain type de l'éléments *View*.
- Un *thème* est un ensemble d'attributs de formatage qu'on peut appliquer à une unité pour toutes les activités d'une application.
  - Par exemple, on peut définir un thème qui met des couleurs spécifiques pour l'ensemble des éléments d'une fenêtre (bordure et fond), définir la taille du texte et les couleurs des menus.
- Créer ses propres *Styles* et *Thèmes* :
  - Créer un fichier '*styles.xml*' avec un nœud '*<ressource>*' dans le répertoire '*res/values*' du projet. Pour chaque style ou thème il faut ajouter un élément '*<style name="nom_de_style">*'. Les éléments de style sont déclarés à l'intérieur par des '*<item name="android:style>valeur</item>*'.