

Présentation du paquet RAD pour Java.

Définition :

RAD signifie Rapid Application Development.

Le but du paquet RAD est de permettre d'écrire une application en Java.

Ce n'est pas un générateur de code, le paquet RAD fournit des classes utiles au développement de l'application.

Objectif :

Le but du paquet RAD est de permettre d'isoler le code métier d'une application.

Le modèle de l'application sera alors en deux couches :

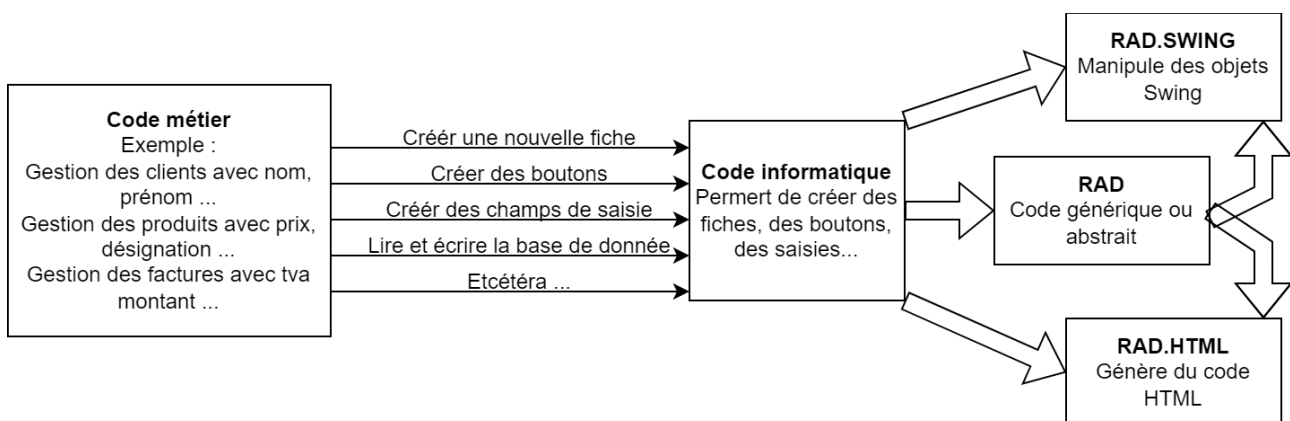
- La couche métier.
- La couche application informatique.

Le paquet RAD facilite l'écriture de la couche application informatique. Le code d'une application qui utilise le paquet RAD sera donc en grande majorité composé de code métier. C'est pour cette raison que le développement de l'application est rapide. Sa maintenance en est aussi facilitée.

Le paquet RAD permet d'écrire une application de type Desktop ou une application Web ou une application à la fois Desktop et Web.

L'application Desktop utilise Swing, l'application Web utilise HTML.

Voici un schéma d'une application RAD :



Applications du paquet RAD :

Le paquet RAD inclut un accès à une base de donnée. Il est orienté création d'application de base de donnée, mais pas obligatoirement.

La parité Swing/Html n'est pas forcément respectée. On peut donc avoir des fonctionnalités uniquement Swing ou uniquement HTML.

Il existe des possibilités de manipulation graphique dans la partie HTML.

Des ouvertures sont possible pour créer ses propres composants ou alors créer de nouveaux composants RAD.

L'application HTML est légère et optimisée. Elle est donc très rapide à l'exécution. Son mode de fonctionnement est particulier.

Présentation du paquet RAD pour Java.

Puisque cette documentation est longue et que paradoxalement, il faut du temps pour mettre au point le développement rapide d'applications, je commence par des exemples de code.

Ces exemples de codes n'utilisent pas forcément directement le paquet RAD. Il utilisent aussi des classes du projet tel que `ErpFiche`. Ces exemples sont donc du code métier de l'application. Par contre, tous les composants créés sont des composants RAD. Par exemple des boutons RAD sont créés par la méthode `createButton`.

L'important pour le moment n'est pas de comprendre tout ce que fait ce code, mais de comprendre que l'on peut rapidement ajouter, supprimer ou modifier les composants de l'application. Le code est valable pour l'application Desktop et pour l'application Web. Modifier ce code modifiera les deux applications simultanément.

Voici le code source de la classe `FicheMenu` de l'application ERP :

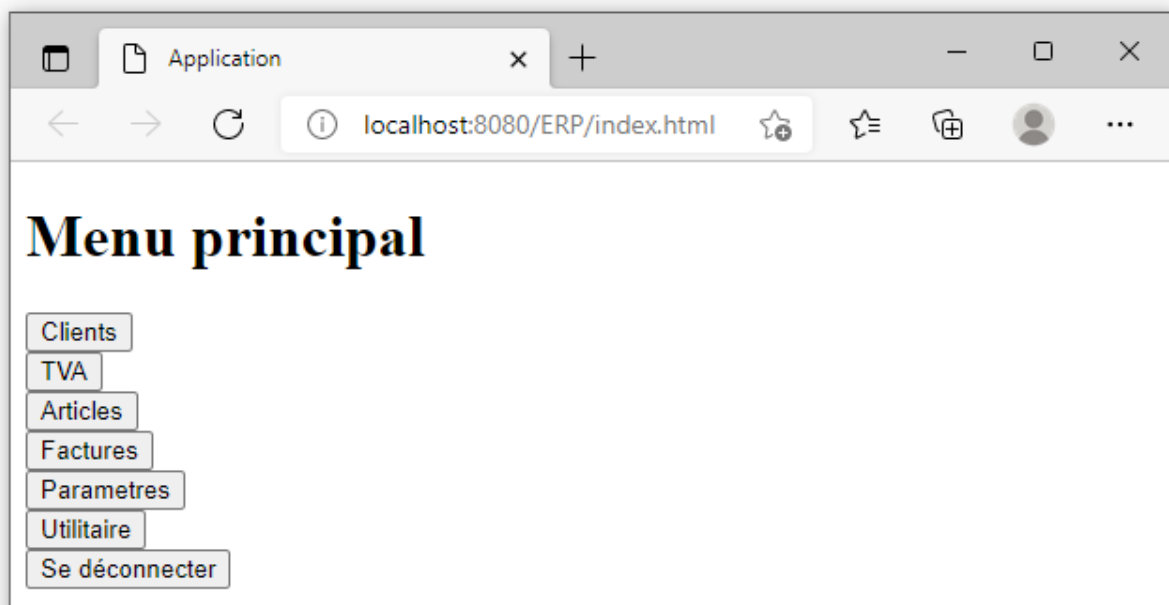
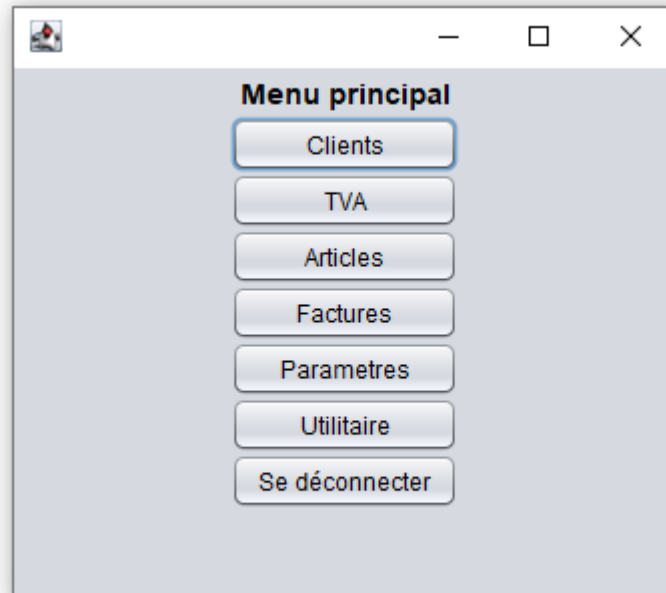
```
public class FicheMenu {
    private ErpFiche erpFiche;
    public FicheMenu(ErpFiche erpFiche) {
        this.erpFiche = erpFiche;
        createContent();
    }
    private void createContent() {
        erpFiche.ajoutTitre("Menu principal");

        erpFiche.ajoutComposant(erpFiche.componentFactory
            .createButton("clients", "Clients").addActionListener(e -> {
                ErpFiche fiche = erpFiche.createFiche("Clients");
                new FicheClients(fiche);
            }));
        erpFiche.ajoutComposant(erpFiche.componentFactory
            .createButton("TVA", "TVA").addActionListener(e -> {
                ErpFiche fiche = erpFiche.createFiche("TVA");
                new FicheTVAs(fiche);
            }));
        erpFiche.ajoutComposant(erpFiche.componentFactory
            .createButton("articles", "Articles").addActionListener(e -> {
                ErpFiche fiche = erpFiche.createFiche("Articles");
                new FicheArticles(fiche);
            }));
        erpFiche.ajoutComposant(erpFiche.componentFactory
            .createButton("factures", "Factures").addActionListener(e -> {
                ErpFiche fiche = erpFiche.createFiche("Factures");
                new FicheFactures(fiche);
            }));
        erpFiche.ajoutComposant(erpFiche.componentFactory
            .createButton("parametres", "Parametres").addActionListener(e -> {
                ErpFiche fiche = erpFiche.createFiche("Parametres");
                new FicheParametres(fiche);
            }));
        erpFiche.ajoutComposant(erpFiche.componentFactory
            .createButton("utils", "Utilitaire").addActionListener(e -> {
                ErpFiche fiche = erpFiche.createFiche("Utilitaire");
                new FicheUtilitaire(fiche);
            }));
        erpFiche.ajoutComposant(erpFiche.componentFactory
            .createButton("Fermer", "Se déconnecter").addActionListener(e -> {
                erpFiche.close();
            }));
    }
}
```

Présentation du paquet RAD pour Java.

Voici le résultat obtenu :

La description du paquet RAD étant technique, il ne faut pas se focaliser sur l'aspect graphique qui est à améliorer.



Présentation du paquet RAD pour Java.

Classe Articles représentant la liste des articles.

```
public class FicheArticles {
    private ErpFiche erpFiche;
    private LogiqueListe logiqueListe;

    public FicheArticles(ErpFiche erpFiche) {
        this.erpFiche = erpFiche;
        if (erpFiche.accesDatabase())
            createContent();
    }

    private void createContent() {
        erpFiche.ajoutTitre("Liste des articles");
        RadSQLGrid grille = createGrille(erpFiche);
        logiqueListe = new LogiqueListe(erpFiche, grille, createfiche->{
            ErpFiche fiche = FicheArticles.this.erpFiche.createFiche("ARTICLE");
            new FicheArticle(fiche, grille.getSelectedLine(),
                grille.getKeyValues(), oncloseKey->{
                    FicheArticles.this.logiqueListe.rafraichit(oncloseKey);
                });
        });
    }

    public static RadSQLGrid createGrille(ErpFiche erpFiche) {
        RadSQLGrid grille = erpFiche.componentFactory
            .createRadSQLGrid("GRARTICLE", ErpFiche.conn, "ID",
                "SELECT ID, DESIGNATION as Désignation , PRIX_UNITAIRE as Prix FROM ARTICLE",
                1, null);
        grille.setGridSearch(createSearch(erpFiche));
        return grille;
    }

    public static SQLGridSearch createSearch(ErpFiche erpFiche) {
        SQLGridSearch gs = new SQLGridSearch(erpFiche.componentFactory) {
            public RadTextField tfRecherche;
            public void init() {
                tfRecherche = createTextField("Recherche");
            }
            public String getFilter(ArrayList<String> values) {
                StringBuilder sql = new StringBuilder();
                String recherche = tfRecherche.getValue();
                if (!"".equals(recherche)) {
                    sql.append("(DESIGNATION LIKE ?)");
                    values.add("%" + recherche + "%");
                }
                return sql.toString();
            }
            public RadContainer getContainer() {
                RadContainer container = createContainer("Recherche");
                container.addLabelComponent("Recherche :", tfRecherche);
                return container;
            }
        };
        return gs;
    }
}
```

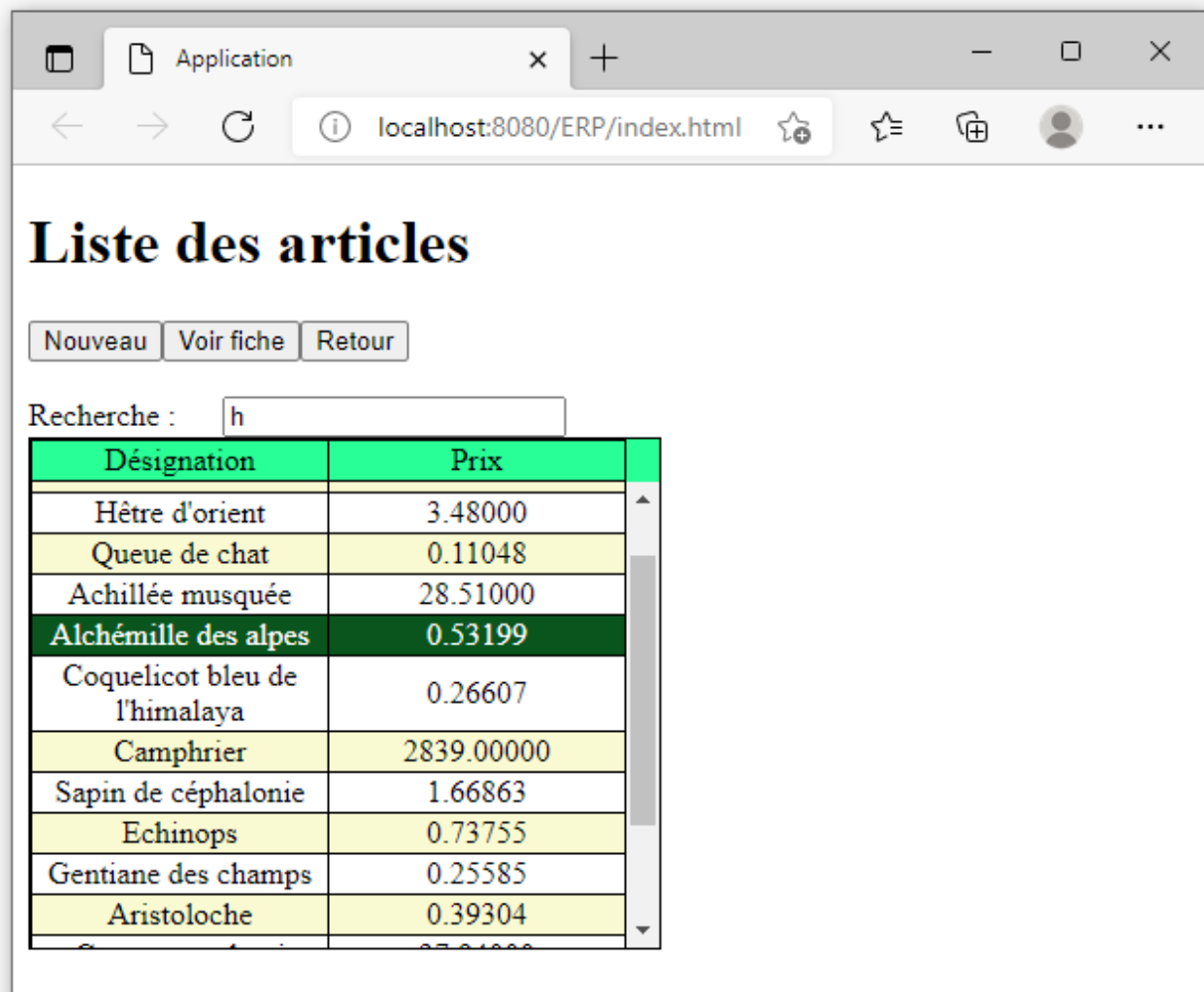
Présentation du paquet RAD pour Java.

Voici le résultat obtenu :



The screenshot shows a Java Swing window titled "Liste des articles". It features three buttons at the top: "Nouveau", "Voir fiche", and "Retour". Below the buttons is a search field labeled "Recherche :" containing the letter "h". The main content is a table with two columns: "Désignation" and "Prix". The table lists various plants and their prices. The row for "Alchémille des alpes" is highlighted in blue.

Désignation	Prix
Petite pervenche	77.00000
Fuchsia	37.52000
Hêtre d'orient	3.48000
Queue de chat	0.11048
Achillée musquée	28.51000
Alchémille des alpes	0.53199
Coquelicot bleu de l'himalaya	0.26607
Camphrier	2839.00000
Sapin de céphalonie	1.66863
Echinops	0.73755
Gentiane des champs	0.25585
Aristoloché	0.39304
Courge spaghetti	37.94000
Orchidée	17854.00000



The screenshot shows a web browser window with a single tab titled "Application". The address bar shows "localhost:8080/ERP/index.html". The page content is identical to the Java Swing window, but the table has a different styling. The table has two columns: "Désignation" and "Prix". The row for "Alchémille des alpes" is highlighted in dark green.

Désignation	Prix
Hêtre d'orient	3.48000
Queue de chat	0.11048
Achillée musquée	28.51000
Alchémille des alpes	0.53199
Coquelicot bleu de l'himalaya	0.26607
Camphrier	2839.00000
Sapin de céphalonie	1.66863
Echinops	0.73755
Gentiane des champs	0.25585
Aristoloché	0.39304

Présentation du paquet RAD pour Java.

Classe Article représentant la fiche d'article.

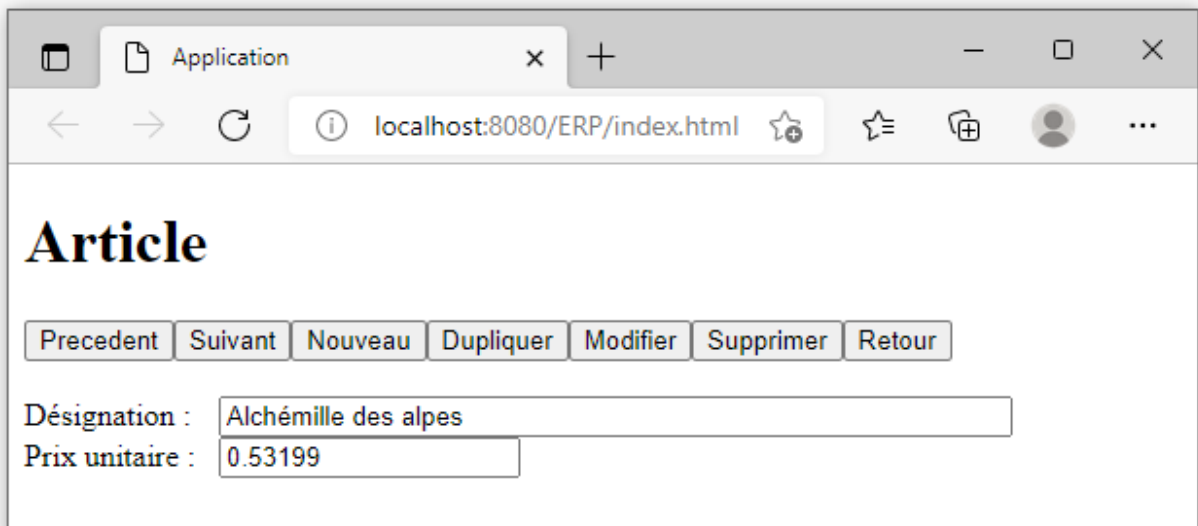
```
public class FicheArticle {
    private ErpFiche erpFiche;
    private DataRAD dataRAD;
    private LogiqueSaisie logiqueSaisie;

    public FicheArticle(ErpFiche erpFiche,
        int selectedLine, ArrayList<Long> keyValues, KeySelectListener onClose) {
        this.erpFiche = erpFiche;
        if (erpFiche.accesDatabase()) {
            dataRAD = erpFiche.createDataRad("ARTICLE", "ID");
            logiqueSaisie = new LogiqueSaisie(erpFiche, dataRAD,
                selectedLine, keyValues, onClose);
            createContent();
            logiqueSaisie.init();
        }
    }

    private void createContent() {
        erpFiche.ajoutTitre("Article");
        erpFiche.ajoutComposant(logiqueSaisie.getBoutonsContainer());
        erpFiche.ajoutSaisie("Désignation :",
            dataRAD.createTextField("DESIGNATION"));
        erpFiche.ajoutSaisie("Prix unitaire :",
            dataRAD.createTextField("PRIX_UNITAIRE"));
    }
}
```



The screenshot shows a Java Swing window titled "Article". At the top, there is a row of seven buttons: "Precedent", "Suivant", "Nouveau", "Dupliquer", "Modifier", "Supprimer", and "Retour". Below the buttons, there are two text input fields. The first is labeled "Désignation :" and contains the text "Alchémille des alpes". The second is labeled "Prix unitaire :" and contains the text "0.53199".



The screenshot shows a web browser window with a single tab titled "Application". The address bar shows "localhost:8080/ERP/index.html". The page content displays the "Article" form, which is identical to the one in the previous screenshot, featuring the same buttons and text fields with the same data.

Présentation du paquet RAD pour Java.

Le paquet RAD fournit des solutions pour générer facilement ces listes et ces fiches de saisies.

Ce document est composé en plusieurs sections :

- | | |
|--------------------------------|---|
| - Classe DataRAD | Gestion des données de la base. |
| - Composant RadSQLGrid | Afficher le résultat d'une requête SQL dans une grille. |
| - Classe RadComponentFactory | Pour que l'application soit Web et Desktop. |
| - Classe WebService | Communication de l'application Web. |
| - Application Desktop ou Web | Structure de l'application. |
| - Application Desktop | Structure de l'application Desktop. |
| - Application Web / Servlet | Structure de l'application Web. |
| - Application Web / Navigateur | L'application coté navigateur avec JavaScript. |
| - Classe ErpFicheHtml | Structure de l'application Web. |
| - Améliorations | Toujours plus |
| - Annexes | Tout le détail public du paquet RAD. |
| - Pensées philosophiques | Penser RAD |

Classe DataRAD

La classe DataRAD classe sert à réaliser le CRUD (Créate Read Update Delete). Elle génère des requêtes SQL en pour lire et afficher des données et aussi pour sauver de manière sécurisée les données saisie.

Constructeur

En prérequis, il faut un objet Connection comme par exemple :

```
connection = DriverManager.getConnection("jdbc:mysql://localhost:3306/ [...])
```

Cette classe fonctionne en sélectionnant une table de la base de donnée. Son constructeur a donc besoin de la connexion à la base de donnée et du nom de la table à utiliser. On précise également le nom du champ clé. Par exemple la table "CLIENT" dont l'enregistrement est identifié par "ID".

La classe DataRAD est abstraite. Pour pouvoir l'utiliser il faut instancier une de ses classe dérivée. Voici comment on peut instancier un objet DataRad pour une table "CLIENT":

- En DataSwing pour les applications Desktop :

```
DataRAD dataClient = new DataSwing(connection, "CLIENT", "ID");
```

- En DataHtml pour les applications Web :

```
DataRAD dataClient = new DataHtml(webService, connection, "CLIENT", "ID");
```

La variable webService sera expliquée plus tard.

Les champs de la table et le CRUD :

Lorsque l'objet dataRAD est instancié, on va détailler les champs de la table pour pouvoir les utiliser. La façon la plus simple est par la méthode createField:

```
public DataFieldRAD createField(String fieldName)
```

Exemple :

```
DataFieldRAD nomClient = dataClient.createField("NOM");
```

A ce stade, on peut déjà utiliser le CRUD. Par exemple voici comment on peut créer, lire, modifier supprimer des données :

Exemple :

```
DataRAD dataClient = new DataSwing(connection, "CLIENT", "ID");
DataFieldRAD nomClient = dataClient.createField("NOM");
nomClient.setValue("Dupont");
dataClient.create();
nomClient.setValue("Durand");
dataClient.create();
dataClient.read(1L);
System.out.println(nomClient.getValue());
nomClient.setValue("Dupond");
dataClient.update();
dataClient.delete();
dataClient.delete(2L);
```


Classe DataRAD

Associer les champ de la table aux champs de saisie :

Lorsque l'objet dataRAD est instancié, on peut associer un champ de la table à un champ de saisie. La méthode la plus simple est d'utiliser la méthode createDataField:

```
public void createDataField(String fieldName, DataFieldRAD dataField)
```

Le type DataFieldRAD est une interface qui regroupe l'ensemble des champs de saisie que l'on peut lier a un objet dataRAD. Ces champs de saisie sont par exemple une saisie de texte, une saisie d'une liste déroulante, un RadioGroup ou autre ...

Le composant de saisie le plus simple est une saisie de texte représentée par la classe RadTextField. Par exemple voici comment on peut créer une zone de saisie pour le champ nom :

```
DataSwing dataClient = new DataSwing(connection, "CLIENT", "ID");
SwingTextField nomClient = new SwingTextField(10);
dataClient.createDataField("NOM", nomClient);
JPanel jpane = new JPanel();
jpane.add(nomClient.getComponent());
```

La méthode getComponent() renvoie l'objet swing jTextField qui à été généré.

Il est possible d'écrire cela de manière plus courte puisque l'objet dataRAD peut lui même générer un champ de saisie. On utilise alors la méthode createTextField. Exemple :

```
DataSwing dataClient = new DataSwing(connection, "CLIENT", "ID");
SwingTextField nomClient = dataClient.createTextField("NOM");
JPanel jpane = new JPanel();
jpane.add(nomClient.getComponent());
```

Encore plus réduit :

```
DataSwing dataClient = new DataSwing(connection, "CLIENT", "ID");
JPanel jpane = new JPanel();
jpane.add(dataClient.createTextField("NOM").getComponent());
```

Il existe un second intérêt à utiliser la méthode createTexteField pour créer les zones de saisie : La zone de saisie ainsi créée aura une taille adaptée au champ de la base de donnée.

Lorsque les champs de saisies sont créés et posés sur la fiche, les méthodes du CRUD utiliseront ces zones de textes pour lire ou sauvegarder les données. La méthode read() remplira les zones de textes avec les données de la base. Les méthodes create() et update() utiliseront les données saisies dans ces zones pour sauver les données.

Voici la liste des zones de saisie que peut générer la classe DataRad :

```
RadStaticText createStaticText(String fieldName);
RadTextField createTextField(String fieldName);
RadRadioGroup createRadioGroup(String fieldName, String[] values, String[] labels);
RadComboBox createSQLComboBox(String fieldName, String sql);
RadButtonField createButtonField(String fieldName, String buttonlabel);
RadTextField createDateField(String fieldName);
```

Classe DataRAD

Accès aux données :

Il est possible d'accéder aux données des champs lus par la méthode read() ou aux champs de saisie associés. On peut écrire dans ces champs grâce à la méthode setFieldValue.

Par exemple :

```
SwingTextField nomClient = dataClient.createTextField("NOM");
jpane.add(nomClient.getComponent());
nomClient.setValue("Durand");
System.out.println(nomClient.getValue());
```

Est équivalent à :

```
jpane.add(dataClient.createTextField("NOM").getComponent());
dataClient.setFieldValue("NOM", "Durand");
System.out.println(dataClient.getFieldValue("NOM"));
```

Note : Cette méthode est moins efficace mais elle est plus lisible. Il n'est plus utile de conserver une référence à la zone de saisie créée puisque l'objet dataRad peut la retrouver.

Divers :

Voici les méthodes qui donnent accès au nom de la table et à la valeur de la clé :

```
public String getTableName()
public Long getKeyValue()
public void setKeyValue(Long keyValue)
```

La méthode clear efface tous les zones de saisie:

```
public void clear()
```

La méthode setReadOnly() fait passer les zones de saisie en lecture seule ou pas.

```
public void setReadOnly(boolean readonly)
```

Classe DataRAD

Les DataRAD liés:

Tout comme il est possible de lier des tables d'une base de donnée, on peut lier des dataRads. Par exemple une table commande peut être liée à la table client par le champ ID_CLIENT.

On pourrait alors obtenir le nom du client de la commande 1 par la requête SQL suivante :

```
SELECT CLIENT.NOM
FROM COMMANDE
LEFT JOIN CLIENT
ON CLIENT.ID=COMMANDE.ID_CLIENT
WHERE COMMANDE.ID=1;
```

Similairement, on pourra lier deux objets dataRad par la méthode createLink:

```
public void createLink(String fieldName, DataRAD dr)
```

Exemple :

```
DataSwing dataCommande = new DataSwing(connection, "COMMANDE", "ID");
DataSwing dataClient = new DataSwing(connection, "CLIENT", "ID");
dataCommande.createLink("ID_CLIENT", dataClient);
dataClient.createField("NOM");
dataCommande.read(1L);
System.out.println(dataClient.getFieldValue("NOM"));
```

Il est possible d'écrire cela de manière plus courte puisque l'objet dataRAD peut lui même générer un dataRAD lié en utilisant la méthode createLinkData:

```
DataRAD dataCommande = new DataSwing(connection, "COMMANDE", "ID");
DataRAD dataClient = dataCommande.createLinkData("ID_CLIENT","CLIENT","ID");
dataClient.createField("NOM");
dataCommande.read(1L);
System.out.println(dataClient.getFieldValue("NOM"));
```

Lorsque les dataRAD sont liés, le fait d'utiliser la méthode read sur un objet va appeler en cascade la méthode read des objets dataRAD liés. Il y a donc deux requêtes SQL successives qui sont exécutées lorsque dataComande.read() est appelée. Cet appel va faire un appel interne à dataClient.read() puisque dataClient est lié.

Les événements :

Il existe deux événements gérés par la classe DataRAD :

```
public DataRAD addRefresh(KeySelectListener ksl)
```

L'évènement Refresh est déclenché par les méthodes read(), clear(), après delete, après update ou après create(). On peut l'utiliser par exemple pour afficher un calcul.

```
public DataRAD addValidateSaveListener(ListenerBooleanRAD vl)
```

L'évènement ValidateSave est déclenché avant un appel à update() ou à create(). Si false est renvoyé, alors la sauvegarde est annulée. Il est possible d'exploiter cet événement pour stocker des résultats de calculs dans la base ou pour confirmer la validité de la saisie avant de sauvegarder.

Composant RadSQLGrid

Le composant RADSQLGrid est un composant visuel qui permet d'afficher, sous forme d'une grille, des données en provenance de la base de donnée. Il peut contenir des colonnes calculées et être accompagné de zones de saisies permettant de faire des recherches sur cette grille.

Le composant RADSQLGrid :

Le composant RADSQLGrid est représenté par l'interface RADSQLGrid.

Constructeur

En prérequis, il faut un objet *Connection* comme par exemple :

```
connection = DriverManager.getConnection("jdbc:mysql://localhost:3306/ [...]")
```

Ce composant fonctionne à partir d'une requête SQL. Son constructeur a donc besoin de la connexion à la base de donnée (*Connection conn*) et d'une requête SQL sous forme de *String* (*String sql*). On précise également le nom du champ clé pour d'autres fonctionnalités (*String KeyField*). Ce champ clé doit obligatoirement être présent dans la requête. Il est possible de cacher (et de ne pas générer) les colonnes des *n* premiers champs de la requête. Pour cela il faut préciser par le paramètre *hiddenFieldsCount* (*int hiddenFieldsCount*) le nombre de champs à cacher. Habituellement, il sera à 1 puisque l'on va cacher l'identifiant que l'on placera en début de requête. On pourra aussi passer en paramètre un tableau de champs calculés qui seront insérés dans la grille.

Le composant RADSQLGrid est une interface donc pour pouvoir l'utiliser il faut instancier une de ses classe dérivée qui sont *SwingSQLGrid* ou *HtmlSQLGrid*. Voici les constructeurs de ces classes :

- En Swing pour les applications Desktop :

```
public SwingSQLGrid(Connection conn, String KeyField, String sql,
                    int hiddenFieldsCount, GridCalculatedField[] calculatedFields)
```
- En Html pour les applications Web :

```
public HtmlSQLGrid(String name, Connection conn, String KeyField, String sql,
                    int hiddenFieldsCount, GridCalculatedField[] calculatedFields)
```

Les composants HTML ont tous besoin d'un nom, la variable name sera expliquée plus tard.

Exemple d'une grille qui affiche le nom et le prénom de la table "CLIENT" :

```
RadSQLGrid grille = new SwingSQLGrid(connection,"ID",
    "SELECT ID,NOM as Nom,PRENOM as Prénom FROM CLIENT",
    1, null);
```

Dans cet exemple on utilise l'instruction SQL *as* pour pouvoir modifier les titres de colonnes de la grille. Les noms de colonnes affichés seront les noms des colonnes SQL. Il est possible d'utiliser les guillemets pour des noms de colonnes à plusieurs mots comme par exemple :

```
"SELECT ID,NOM as 'Nom de naissance',PRENOM as 'Prénom usuel' FROM CLIENT"
```

Composant RadSQLGrid

Les champs calculés avec la classe SQLGridCalculatedField:

Pour réaliser un champ calculé, il faut créer un objet de type SQLGridCalculatedField. Le constructeur demande la position où sera inséré la colonne et le titre à afficher. Cette classe est abstraite, il faut donc l'instancier en utilisant une classe anonyme et compléter la méthode Calculate. La méthode calculate fournit en paramètre le ResultSet rs qui résulte de l'exécution de la requête SQL.

Exemple :

Créer un champ calculé nommé dénomination inséré en début de grille contenant la concaténation du nom et du prénom :

```
SQLGridCalculatedField denomination = new SQLGridCalculatedField(0, "Dénomination") {  
    @Override  
    public String Calculate(ResultSet rs) throws SQLException {  
        return rs.getString("NOM") + " " + rs.getString("PRENOM");  
    }  
};
```

On pourra alors utiliser ce champ calculé comme suit :

```
RadSQLGrid grille = new SwingSQLGrid(connection, "ID",  
    "SELECT ID,NOM,PRENOM FROM CLIENT",  
    3, new SQLGridCalculatedField[] {denomination});
```

Dans cet exemple les 3 champs de la requête sont cachés et le champ dénomination apparaît en colonne 0.

Note :

Fournir une position erronée ou d'autres données erronées provoquera une erreur à l'exécution.

Composant RadSQLGrid

Utilisation de SQLGridSearch

La classe SQLGridSearch sert à générer des zones de recherches liés à une grille. Ces zones de recherches seront fonctionnelles et déclencheront un rafraîchissement de la grille à chaque modification.

Constructeur

La classe SQLGridSearch est abstraite. On ne peut donc pas l'instancier. En effet cette classe est incomplète. Cette classe est prévue pour être instanciée en classe anonyme. On pourra alors compléter les méthodes manquantes pour finaliser la classe. Ces méthodes sont les suivantes :

- `public abstract void init();`
- `public abstract String getFilter(ArrayList<String> values);`
- `public abstract RadContainer getContainer();`
- La méthode `init` est prévue pour la création des zones de recherches.
- La méthode `getFilter` renvoie la partie WHERE de la requête SQL qui correspond à la recherche souhaitée.
- La méthode `getContainer()` construit et remplit le composant `RadContainer` contenant les champs de saisie associés aux recherches.

Le constructeur aura aussi besoin d'un objet `RadComponentFactory`. Cet objet sera utilisé en interne pour créer les zones de saisies. Le fonctionnement de la classe `RadComponentFactory` sera expliqué plus tard.

```
public SQLGridSearch(RadComponentFactory componentFactory)
```

Exemple d'utilisation :

Cet exemple montre comment on peut faire une recherche dans le nom ou le prénom.

```
componentFactory = new SwingComponentFactory();
SQLGridSearch gs = new SQLGridSearch(componentFactory) {
    public RadTextField tfRecherche;

    @Override
    public void init() {
        tfRecherche = createTextField("Recherche");
    }

    @Override
    public String getFilter(ArrayList<String> values) {
        String recherche = tfRecherche.getValue();
        if ("".equals(recherche)) return null;
        values.add("%" + recherche + "%");
        values.add("%" + recherche + "%");
        return "(NOM LIKE ? OR PRENOM LIKE ?)";
    }

    @Override
    public RadContainer getContainer() {
        RadContainer container = createContainer("Recherche");
        container.addLabelComponent("Recherche :", tfRecherche);
        return container;
    }
};
```

Classe RadComponentFactory

La classe RadComponentFactory est un générateur de composants RAD. La classe SQLGridSearch l'utilise, mais pour le reste de l'application on pourrait très bien s'en passer.

En effet, Il est possible d'instancier les composants RAD directement. Par exemple pour créer un nouveau bouton on peut très bien faire comme suit :

Pour une application Desktop :

```
RadButton monBouttonSwing = new SwingButton("un bouton");
```

Pour une application Web

```
RadButton monBouttonHtml = new HtmlButton("button", "Un bouton");
```

Si l'on veut écrire du code qui soit valable pour une application Desktop et une application Web, il est alors utile d'utiliser la classe RadComponentFactory.

```
private void creeInterface(RadComponentFactory componentFactory) {  
    RadButton monBoutton = componentFactory.createButton(  
        "button", "Un bouton");  
    // reste du code  
}
```

Il sera alors possible d'utiliser cette méthode dans une application Desktop et une application Web. On emploiera alors cette méthode dans chacune des deux applications :

Pour une application Desktop :

```
creeInterface(new SwingComponentFactory());
```

Pour une application Web

```
WebService webService = new WebService("./ApplicationRAD", "interface");  
creeInterface(new HtmlComponentFactory(webService));
```

Classe WebService

La classe WebService est utile uniquement aux applications Web. Les applications Desktop n'en ont pas besoin. Quand un composant Desktop est posé dans l'application (en l'ajoutant dans un JPanel) il va fonctionner tout seul avec Swing. Les événements vont fonctionner. Quand on modifie une propriété le résultat sera immédiatement visible à l'écran.

Pour les application Web, les événements vont devoir circuler du navigateur jusqu'au serveur. Les données saisies dans les zones de saisies doivent aussi être transmises au serveur. Ensuite toutes les propriétés de chaque composant qui ont été modifiées doivent être retransmises vers le navigateur Web. Charge ensuite au navigateur d'appliquer ces modifications.

Constructeur

Un WebService a besoin de deux paramètres :

- Le nom de l'API (String `apiUrl`). Cette URL sera celle de la Servlet de l'application. Il est possible d'utiliser plusieurs Servlet mais une seule suffit pour toute l'application.
- Son nom (String `name`). Il est possible d'utiliser plusieurs WebService dans une seule page mais généralement, un seul suffit. Son nom sert à le différencier dans le DOM côté navigateur. Son nom sera ajouté en préfixe à tous les composants RAD qui lui seront associés.

```
public WebService(String apiUrl, String name) {
```

Exemple :

```
WebService webService = new WebService("./ApplicationRAD","interface");
```

Remarque :

Dans le code métier de l'application, la classe WebService devrait être invisible. Il sera employé dans la couche code informatique de l'application.

Association aux composants

Pour qu'un composant soit pris en compte par un WebService, il faut l'associer. Pour cela on peut utiliser la méthode `addComponent` :

Par exemple :

```
WebService webService = new WebService("./ApplicationRAD","interface");
HtmlButton monBouttonHtml = new HtmlButton("button","Un bouton");
webService.addComponent(monBouttonHtml);
```

Note :

Les classes `DataHtml`, `HtmlComponentFactory` et `SQLGridSearch` possèdent un `WebService`. Tous les composants créés par ces moyens là n'ont donc pas besoin d'utiliser la méthode `addComponent`. L'appel à `addComponent` est implicite lors de l'instanciation de nouveaux composants de saisie par ces classes là.

Important :

Un composant ne peut être associé qu'à un seul Web Service. Associer un composant qui est déjà associé ne modifiera pas son association. Il est toute fois possible, par sécurité et confort dans le code, d'associer plusieurs fois un composant pour être sûr qu'il est bien associé.

A voir ultérieurement :

```
public void setValues(Map<String, String[]> values) {
public void writeResponses(PrintWriter out) throws IOException {
```


Application Desktop ou Web

Cette couche informatique de l'application est à écrire et ne fait pas partie du paquet rad. Pour l'application ERP, j'ai utilisé une classe ErpFiche qui est abstraite de manière à pouvoir être utilisée par les classe du code métier qui seront valable en Html et en Swing.

Cette classe comporte (de manière simplifiée ici) : le code de connexion à la base de donnée ainsi qu'une série de méthodes abstraites. Il contient aussi le code qui permet de faire une sélection parmi une liste. Ce code aurait put faire partie du paquet RAD mais le choix de la mise en page fait qu'il n'est pas de nature à être générique.

```
public abstract class ErpFiche {
    public static Connection conn = null;
    protected ErpFiche ficheSuivante;
    protected ErpFiche fichePrecedente;
    protected RadComponentFactory componentFactory;
    public static String canClose = null;

    protected abstract void ajoutTitre(String titre);
    protected abstract ErpFiche createFiche(String nom);
    protected abstract void ajoutComposant(RadComponent comp);
    protected abstract void ajoutGrille(RadSQLGrid grille);
    protected abstract void ajoutSaisie(String label, RadComponent component);
    protected abstract void showMessage(String message);
    protected abstract void afficheFichier(String nomFichier);
    protected abstract void close();
    protected abstract DataRAD createDataRad(String tableName, String keyField);

    public ErpFiche getFicheSuivante() {
        return ficheSuivante;
    }
    public void setFicheSuivante(ErpFiche ficheSuivante) {
        this.ficheSuivante = ficheSuivante;
    }

    public static boolean establish() {
        if (conn == null) {
            try {
                Class.forName("com.mysql.jdbc.Driver");
                conn = DriverManager.getConnection(
                    "jdbc:mysql://localhost:3306/erp?
                    allowPublicKeyRetrieval=true&useSSL=false
                    &useLegacyDatetimeCode=false&serverTimezone=Europe/Paris",
                    "erpuser", "erppass");
            } catch ...
        }
        return true;
    }

    public boolean accesDatabase() {...}
    public static String getCanClose() {...}
    public static void setCanClose(String canClose) {...}
    protected void createFicheSelectionGrille(ErpFiche ficheSelectionGrille,
        String titre, DataFieldRAD dataField,
        RadSQLGrid grille, KeySelectListener onSelection) {...}
    public static String concateneStringAvecEspace(String s1, String s2) {...}
}
```

Application Desktop

La partie Desktop utilise la classe `ErpFicheSwing`. Cette classe consiste surtout à construire les composants `RadComponentFactory` et `DataRad` en leur version Swing. Le reste est de la mise en page en utilisant Swing. Pour la documentation j'ai retiré du code de manière à ne laisser que ce qui est essentiel à la compréhension du paquet RAD. Consulter le projet pour le source complet.

```
public class ErpFicheSwing extends ErpFiche {
    protected static Erp JFrame;
    private GridBagConstraints gbc;
    protected JComponent panelNorth;
    protected JComponent panelCenter;

    public ErpFicheSwing() {
        super();
        componentFactory = new SwingComponentFactory();
        panelNorth = new JPanel();
        panelCenter = new JPanel();
        (reste de l'initialisation)
    }

    protected DataRad createDataRad(String tableName, String keyField) {
        DataSwing data = new DataSwing(conn, tableName, keyField);
        return data;
    }

    protected ErpFicheSwing createFiche(String nom) {
        ErpFicheSwing erpFicheSwing = new ErpFicheSwing();
        ficheSuivante = erpFicheSwing;
        erpFicheSwing.fichePrecedente = this;
        (échange des pannels affichés avec les pannels de la fiche créé)
        return erpFicheSwing;
    }

    protected void close() {
        (échange des pannels affichés avec les pannels de la fiche précédente)
    }

    protected void ajoutTitre(String titre) {
        panelNorth.add(new JLabel(titre), gbc);
        (manipuler gbc, changer de font)
    }

    protected void ajoutComposant(RadComponent comp) {
        panelNorth.add(((SwingComponent) comp).getComponent(), gbc);
        (manipuler gbc)
    }

    protected void ajoutSaisie(String label, RadComponent component) {
        panelNorth.add(new JLabel(label), gbc);
        panelNorth.add(((SwingComponent) component).getComponent(), gbc);
        (manipuler gbc)
    }

    protected void ajoutGrille(RadSQLGrid grille) {
        panelCenter = ((SwingSQLGrid) grille).getScrollPane();
        Component searchPane = ((SwingSQLGrid) grille).getSearchPane();
        (placer les deux panel dans la fiche)
    }

    protected void showMessage(String message) {
        JOptionPane.showMessageDialog(null, message);
    }

    protected void afficheFichier(String nomFichier) (afficher le fichier à l'écran)
}
```

Application Desktop

La classe de démarrage :

Pour la documentation j'ai retiré du code de manière à ne laisser que ce qui est essentiel à la compréhension du paquet RAD. Consulter le projet pour le source complet.

La manière de construire une fiche est spéciale. On instancie d'abord un objet `ErpFicheSwing` et ensuite on construit un objet de type `FicheLogin`. La référence à `FicheLogin` n'est pas gardée puisque on utilise l'objet `ErpFicheSwing` pour la suite.

```
public class Erp extends JFrame {
    private static final long serialVersionUID = 1L;
    static JPanel mainPanel;

    public Erp() throws HeadlessException {
        super();
        this.setSize(1000, 800);
        this.setLocationRelativeTo(null);
    }

    public static void main(String[] args) throws UnsupportedLookAndFeelException {
        Erp erp = new Erp();
        erp.createApplication();
        erp.setVisible(true);
    }

    private void createApplication() {
        mainPanel = new JPanel();
        mainPanel.setLayout(new BorderLayout());
        setContentPane(mainPanel);
        DataRAD.maxFieldLength = 30;
        ErpFicheSwing premiereFiche;
        premiereFiche = new ErpFicheSwing();
        new FicheLogin(premiereFiche);
        mainPanel.add(premiereFiche.panelCenter, BorderLayout.CENTER);
        mainPanel.add(premiereFiche.panelNorth, BorderLayout.NORTH);
    }
}
```

Application Web / Servlet

L'application Web se déroule dans une page Web tel que index.html.
Voici par exemple le fichier index.html de l'application ERP :

```
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Application</title>
<link rel="stylesheet" href="css/rad.css?version=1">
</head>
<body>
    <div id="idRadApplication">
    </div>
    <script src="./js/rad.js?version=1"></script>
    <script src="./js/erp.js?version=1"></script>
</body>
</html>
```

Cette page Html est complètement vide sauf une DIV identifiée par idRadApplication. L'intégralité de l'application Web sera construite dans cette DIV. Le script JavaScript rad.js s'occupe de tout le reste. Cette application utilise un second script qui étend le script rad.js, c'est le script erp.js. Ce second script permet une fonctionnalité non prévue par le paquet RAD : Demander la confirmation pour fermer la fenêtre quand la saisie est en cours. Il doit être placé après le script rad.js.

Le script rad.js va utiliser la Servlet de l'application.

Servlet de l'application.

La Servlet ApplicationRadServlet comporte deux parties :

La méthode doGet

La méthode doGet renvoie le code HTML de la fiche en cours. Ce code HTML est généré par la méthode getContent() de la classe ErpFicheHtml. La méthode getFiche(request) permet de renvoyer la fiche en cours de l'application. Cette fiche est stockée dans la Session. Une fiche login est créée par défaut.

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    response.setContentType("text/html");
    response.setCharacterEncoding("UTF-8");
    response.getWriter().print(getFiche(request).getContent());
}
protected ErpFicheHtml getFiche(HttpServletRequest request) {
    HttpSession session = request.getSession();
    ErpFicheHtml ficheHtml = (ErpFicheHtml) session.getAttribute("ficheHtml");
    if (ficheHtml == null) {
        ficheHtml = new ErpFicheHtml(apiURL, "Login");
        new FicheLogin(ficheHtml);
    }
    return ficheHtml;
}
```

Application Web / Servlet

La méthode doPost

La méthode doPost va permettre le dialogue entre le navigateur du client et un Webservice de l'application. Elle va recevoir en paramètres tout ce qui a été saisi dans les zones de saisies liés au Webservice ainsi que la description de l'événement qui l'a déclenché.

Le résultat de la réponse de la méthode doPost peut prendre deux aspects :

- Un Json détaillant les modifications à apporter du DOM du navigateur.
- Du code HTML à afficher par le navigateur.

Lorsque la réponse est une création de fiche alors la réponse va générer le code HTML de la nouvelle fiche. Ce sera la même réponse que pour la méthode doGet.

Lorsque la fiche entière n'est pas à rafraîchir, mais seulement quelques données sont à actualiser alors un Json est utilisé. C'est le cas à chaque fois que l'on fait une action sur une fiche qui n'ouvre pas de nouvelle fiche. Le Json est généré par la méthode setValuesAndWriteResponse de la classe ErpFicheHtml.

```
protected void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    response.setContentType("text/html");
    response.setCharacterEncoding("UTF-8");
    ErpFicheHtml ficheHtml = getFiche(request);
    ficheHtml = ficheHtml.setValuesAndWriteResponse(
        request.getParameterMap(), response);
    if (ficheHtml!=null) {
        HttpSession session = request.getSession();
        if (ficheHtml.storeSession(session)) {
            session.setAttribute("ficheHtml", ficheHtml);
        } else {
            session.invalidate();
        }
        response.getWriter().print(ficheHtml.getContent());
    }
}
```

Application Web / Navigateur

Le coté navigateur Web

Je vais expliquer par l'exemple ce qui se passe dans le navigateur. L'application se trouve sur l'écran de TVA :



Utilisation de la servlet GET

On a vu que la page index.html est une page vide à l'exception d'une DIV nommée idRadApplication. Je vais relire cette page en utilisant le bouton Actualiser du navigateur Web (ctrl+R ou ctrl+F5). Avant cela, je vais modifier la fonction loadApplication dans le script rad.js.

rad.js (extrait de la fin de fichier)

```
function loadApplication() {  
    apiURL = "./ApplicationRAD";  
    GetAjax(apiURL, function(resultat) {  
        console.log(resultat);  
        var fiche = document.getElementById("idRadApplication");  
        fiche.innerHTML = resultat;  
        doRADInits();  
    })  
}  
loadApplication();
```

Ce script fait une requête Ajax sur l'API `./ApplicationRAD`. Le résultat de cette requête est inséré dans la DIV nommée idRadApplication. Je rajoute `console.log(resultat)` pour pouvoir savoir ce qui est renvoyé par la méthode Get sur la Servlet /ApplicationRAD.

Application Web / Navigateur

Résultat dans la console :

```
<h1>Taux de TVA</h1><div name="boutons" id="idboutons" ><input type="button"
name="TVA:btPrecedent" id="idTVA:btPrecedent" value="Precedent"
onclick="RADEvent('TVA:btPrecedent','./ApplicationRAD','TVA','click')"><input
type="button" name="TVA:btSuivant" id="idTVA:btSuivant" value="Suivant"
onclick="RADEvent('TVA:btSuivant','./ApplicationRAD','TVA','click')"><input
type="button" name="TVA:btNouveau" id="idTVA:btNouveau" value="Nouveau"
onclick="RADEvent('TVA:btNouveau','./ApplicationRAD','TVA','click')"><input
type="button" name="TVA:btDupliquer" id="idTVA:btDupliquer" value="Dupliquer"
onclick="RADEvent('TVA:btDupliquer','./ApplicationRAD','TVA','click')"><input
type="button" name="TVA:btModifier" id="idTVA:btModifier" value="Modifier"
onclick="RADEvent('TVA:btModifier','./ApplicationRAD','TVA','click')"><input
type="button" name="TVA:btSupprimer" id="idTVA:btSupprimer" value="Supprimer"
onclick="if (confirm('Confirmer la suppression ?'))
RADEvent('TVA:btSupprimer','./ApplicationRAD','TVA','click')"><input type="button"
name="TVA:btOK" id="idTVA:btOK" value="OK" class = " hidden"
onclick="RADEvent('TVA:btOK','./ApplicationRAD','TVA','click')"><input type="button"
name="TVA:btCancel" id="idTVA:btCancel" value="Annuler" class = " hidden"
onclick="RADEvent('TVA:btCancel','./ApplicationRAD','TVA','click')"><input
type="button" name="TVA:btRetour" id="idTVA:btRetour" value="Retour"
onclick="RADEvent('TVA:btRetour','./ApplicationRAD','TVA','click')"></div><br><label
for ="idTVA:LIBELLE">Libellé :</label><input name = "TVA:LIBELLE" id =
"idTVA:LIBELLE" value ="TVA normale 20%" maxlength="50" style = "width:30em;" readonly
RADWebServiceValueTVA ><br><label for ="idTVA:TAUX">Taux :</label><input name =
"TVA:TAUX" id = "idTVA:TAUX" value ="20.00" maxlength="5" style = "width:5em;" readonly
RADWebServiceValueTVA ><br>
```

On remarque que l'on obtient le code source HTML de la page. Ce code source à été généré par le paquet RAD il n'est donc pas humainement très lisible. Ce code source ne comporte pas d'entête HTML5 puisqu'il est destiné à être inséré dans la DIV nommée idRadApplication.

Application Web / Navigateur

Utilisation de la servlet POST

Je vais cliquer sur le bouton précédent. Mais avant cela, je vais isoler son code qui est au début du bloc HTML :

```
<input type="button" name="TVA:btPrecedent" id="idTVA:btPrecedent" value="Precedent"
onclick="RADEvent('TVA:btPrecedent','./ApplicationRAD','TVA','click')">
```

L'événement onclick fait appel à la méthode fonction RADEvent.

- En premier paramètre à cet appel, on peut trouver le nom du composant qui déclenche l'événement : 'TVA:btPrecedent'
- En second paramètre on voit l'URL de l'API à utiliser : './ApplicationRAD'
- En troisième paramètre on voit le nom du WebService associé à ce bouton : 'TVA'. En dernier paramètre figure le nom de l'événement : 'click'

Je vais donc modifier la méthode RADEvent du script rad.js

```
function RADEvent(idCaller, apiURL, WebServiceName, evName) {
    apiURL += "?RADEvent=" + encodeURIComponent(idCaller);
    apiURL += "&RADEventName=" + evName
    const requestFieldvalues = document
        .querySelectorAll("[RADWebServiceValue" + WebServiceName + "]");
    for (let element of requestFieldvalues) {
        apiURL += "&" + element.getAttribute("name") + "=" +
            encodeURIComponent(element.value);
    }

    const requestFieldDatavalues = document
        .querySelectorAll("[RADWebServiceDataValue" + WebServiceName + "]");
    for (let element of requestFieldDatavalues) {
        apiURL += "&" + element.getAttribute("name") + "=" +
            encodeURIComponent(element.getAttribute("data-value"));
    }

    RADEventExtras.forEach(RADEventExtra => {
        apiURL += RADEventExtra();
    })

    console.log(apiURL);
    PostAjax(apiURL, function(resultat) {
        console.log(resultat);
        RADFill(resultat)
    })
}
```

Cette fonction fait une requête Ajax par la méthode Post. On remarque que cette fonction a pour but de construire la variable apiURL et de traiter le le résultat de la requête Ajax par la fonction RADFill. Je rajoute donc console.log(apiURL) pour pouvoir voir le détail de l'appel à l'API. Je rajoute aussi console.log(resultat) pour savoir ce qui est retourné.

Application Web / Navigateur

Résultat dans la console :

Voici ce que donne console.log(apiURL)

```
./ApplicationRAD?RADEvent=TVA%3AbtPrecedent&RADEventName=click&TVA:LIBELLE=TVA%20normale%2020%25&TVA:TAUX=20.00
```

On retrouve l'URL de l'API qui est ./ApplicationRAD .

Les paramètres passés sont les suivants (les espaces et caractères spéciaux sont encodés) :

RADEvent = TVA:btPrecedent

RADEventName=click

TVA:LIBELLE=TVA normale 20%

TVA:TAUX=20.00

On remarque que tout est détaillé dans les paramètres : Le nom du composant qui déclenche l'événement, le nom de l'événement ainsi que la valeur de chaque champ de saisie qui est associé au Webservice lié.

En retour à cette requête, on peut observer console.log(resultat) :

```
{ "properties": [ [ "idTVA:btDupliquer", "disabled", false ], [ "idTVA:LIBELLE", "value", "Exonéré de TVA" ], [ "idTVA:TAUX", "value", "0.00" ], [ "idTVA:btSupprimer", "disabled", false ], [ "idTVA:btModifier", "disabled", false ], [ "idTVA:btPrecedent", "disabled", true ], [ "idTVA:btSuivant", "disabled", false ] ], "classes": [ ], "datavalues": [ ], "alerts": [ ], "functions": [ ], "cancelclose": "" }
```

Ce qui est retourné est un JSON.

Il est séparé en plusieurs catégories qui servent à modifier le contenu du navigateur. Ce JSON sera traité en JavaScript par la fonction RADFill.

Remarque :

La fonction RADFill tentera de convertir le résultat de la requête Ajax en JSON. Au cas où elle échoue, le contenu du résultat sera affecté à la DIV idRadApplication. Ceci aura pour effet d'afficher le résultat dans le navigateur.

Il peut y avoir plusieurs raisons à cela :

- Soit le JSON est mal formé. Cela serait dû à une erreur du programme qui génère ce JSON.
- Soit il s'est produit une erreur dans le serveur et Tomcat renvoie une page HTML d'erreur. Cette page sera donc affichée et lisible.
- Soit il y a un changement de fiche dans l'application, et le résultat sera le code HTML à de la nouvelle page à afficher. Cela serait un fonctionnement normal.

Application Web / Navigateur

JSON RAD :

```
{
  "properties": [
    [IDComposant1, NomPropriété1, ValeurPropriété1],
    [IDComposant2, NomPropriété2, ValeurPropriété2],
    [IDComposant3, NomPropriété3, ValeurPropriété3]
  ],
  "classes": [
    [IDComposant1, NomClasseCSS, AjoutOuRetrait],
    [IDComposant2, NomClasseCSS, AjoutOuRetrait]
  ],
  "datavalues": [
    [IDComposant1, DataValue1],
    [IDComposant2, DataValue2],
    [IDComposant3, DataValue3]
  ],
  "alerts": [Alerte1, Alerte2, Alerte3],
  "functions": [
    [Fonction1, Paramètre1],
    [Fonction2, Paramètre2],
    [Fonction3, Paramètre3]
  ]
}
```

Les catégories du JSON sont les suivantes :

– propriétés :

Cette catégorie contient la liste des propriétés à modifier. Elle est composée d'un tableau contenant des tableaux contenant l'identifiant de l'élément du DOM à modifier, le nom de sa propriété à modifier et la valeur à lui appliquer.

```
[[IDComposant1, NomPropriété1, ValeurPropriété1],
 [IDComposant2, NomPropriété2, ValeurPropriété2],
 [IDComposant3, NomPropriété3, ValeurPropriété3]]
```

– classes :

Cette catégorie contient la liste des classes CSS à ajouter ou retirer. Elle est composée d'un tableau de tableau contenant l'identifiant de l'élément, le nom de la classe CSS et un booléen qui est à vrai en cas d'ajout et à faux en cas de retrait.

```
[[IDComposant1, NomClasseCSS, AjoutOuRetrait],
 [IDComposant2, NomClasseCSS, AjoutOuRetrait]]
```

– datavalues

Cette catégorie contient la liste des datavalues à modifier. Pour les composants Web qui ne disposent pas de la propriété value, l'attribut de donnée data-value est utilisé. Il est modifié par ce moyen la.

```
[[IDComposant1, DataValue1], [IDComposant2, DataValue2], [IDComposant3, DataValue3]]
```

– alerts

Ce tableau contient la liste des alertes à afficher.

```
[Alerte1, Alerte2, Alerte3]
```

– functions

Ce tableau contient des tableaux contenant un nom de fonction JavaScript à appeler ainsi qu'un paramètre.

```
[[Fonction1, Paramètre1], [Fonction2, Paramètre2], [Fonction3, Paramètre3]]
```

– canclose

Cette dernière catégorie ne fait pas partie du paquet RAD. Elle a été rajoutée par l'application ERP. Elle permet d'empêcher la fermeture du navigateur alors que la saisie est en cours.

Application Web / Navigateur

Recevoir plus de données : personnalisation du JSON RAD

Il est possible de rajouter des informations supplémentaire dans le JSON RAD. Par exemple l'application ERP ajoute une catégorie "cancelclose". Cette catégorie sera complètement ignorée par la fonction RADFill du script rad.js. En effet, RADFill ne s'occupe que des catégories RAD du JSON.

Pour pouvoir exploiter cette nouvelle catégorie, il faut en faire la demande.

Pour cela, il faut utiliser la fonction RegisterRADFillExtra.

En paramètre de cette fonction, il faut fournir une fonction qui traite le JSON.

Exemple pour exploiter la catégorie cancelclose :

Dans le script erp.js :

```
var cancelclose = "";
RegisterRADFillExtra(function(json) {
    cancelclose = json.cancelclose;
});
```

A chaque réception d'un nouveau JSON RAD, la fonction RADFill appellera cette fonction qui va traiter le JSON. Celle ci se contente de mettre à jour une variable interne utilisée plus loin dans le script.

Envoyer plus de données : Compléter apiURL

Il est aussi possible de transmettre plus de données à la Servlet.

Dans le script rad.js, La fonction RADEvent se charge de faire la requête post vers la Servlet du serveur. Elle construit l'URL sans sa variable interne apiURL en y ajoutant les paramètres à transmettre à la Servlet. A ce moment là, il est possible de fournir des paramètres supplémentaires. Pour cela, il faut utiliser la fonction RegisterRADEventExtra

Exemple de rajout d'un paramètre (interfaceValue):

```
var interfaceValue = "";
RegisterRADEventExtra(function() {
    var result = "&interfaceValue=" + encodeURIComponent(interfaceValue);
    interfaceValue = "";
    return result;
});
```

A chaque nouvelle requête vers la Servlet, fonction RADEvent appellera cette fonction qui va retourner les données à ajouter à la requête. Celle ci par exemple se contente d'ajouter une variable interne utilisée plus loin dans le script. L'appel à encodeURIComponent permet de transmettre les espaces et caractères spéciaux.

Classe ErpFicheHtml

La partie WEB de l'application utilise la classe ErpFicheHtml. Cette classe, comme pour ErpFicheSwing, consiste à construire les composants RadComponentFactory et DataRad, cette fois dans leur version Html. Elle implémente aussi les méthodes qui permettent la mise en page.

Le but final de la classe ErpFicheHtml est de pouvoir fournir le contenu HTML de la page en cours à la Servlet GET et de pouvoir fournir le JSON RAD à la Servlet POST.

Code fondamental

En premier lieu le code fondamental de la classe : Le constructeur qui instancie un Webservice et un ComponentFactory. La méthode de création d'un DataRad est simple. La création d'une fiche et sa fermeture se fait en mettant à jour les variables ficheSuivante et fichePrécédente.

```
public class ErpFicheHtml extends ErpFiche {
    private LinkedList<Object> content;
    private Webservice webService;
    private HttpSession session;
    public ErpFicheHtml(String apiURL, String name) {
        webService = new Webservice(apiURL, name);
        componentFactory = new HtmlComponentFactory(webService);
        content = new LinkedList<Object>();
    }
    @Override
    protected DataRad createDataRad(String tableName, String keyField) {
        DataHtml data = new DataHtml(webService,conn, tableName, keyField);
        return data;
    }
    @Override
    protected ErpFiche createFiche(String nom) {
        ficheSuivante = new ErpFicheHtml(webService.getApiUrl(),nom);
        ficheSuivante.fichePrecedente = this;
        return ficheSuivante;
    }
    @Override
    protected void close() {
        ficheSuivante = fichePrecedente;
        ficheSuivante.ficheSuivante = null;
    }
    public boolean storeSession(HttpSession session) {
        this.session = session;
        return (fichePrecedente != null);
    }
    (...suite du code ...)
}
```

La méthode storeSession indique à la Servlet si elle doit sauvegarder la fiche dans la session. C'est toujours le cas sauf pour la fiche login. Cette fiche login qui est la première fiche de l'application ne possède pas de fiche précédente. Au passage, la session est conservée pour pouvoir être utilisée par la méthode afficheFicher.

Classe ErpFicheHtml

La mise en page

Le contenu de la fiche est stocké dans la variable content (**private** LinkedList<Object> **content**).

Cette méthode permet de garder les composants RAD utilisés dans la fiche et de pouvoir produire le code HTML au dernier moment. De cette manière, si les propriétés des composants sont modifiés après avoir été insérés dans la fiche, alors la génération du code HTML en tiendra compte.

Le type de la variable content est LinkedList<Object>. C'est donc une liste. On peut y ajouter des chaînes de caractères qui seront du code HTML ou des objet de type HtmlComponent.

Voici donc les fonction de mise en page :

```
public class ErpFicheHtml extends ErpFiche {
    (...suite du code ...)

    @Override
    protected void ajoutTitre(String titre) {
        content.add("<h1>" + titre + "</h1>");
    }

    @Override
    protected void ajoutComposant(RadComponent comp) {
        content.add(comp);
    }

    @Override
    protected void ajoutSaisie(String label, RadComponent component) {
        content.add("<label for =\"id\" + ((HtmlComponent)component).getName() + \">\" + label + "</label>");
        content.add(component);
    }

    @Override
    protected void ajoutGrille(RadSQLGrid grille) {
        ajoutComposant(grille);
    }
    (...suite du code ...)
}
```

La méthode getContent()

Cette méthode est utilisée par la Servlet GET et produit le contenu HTML de la fiche.

Elle va insérer le code HTML des composants ou strings stockés dans la variable content.

```
public String getContent() {
    StringBuilder fiche = new StringBuilder();
    for (Object object : content) {
        if (object instanceof String) {
            fiche.append((String)object);
        }
        if (object instanceof HtmlComponent) {
            fiche.append(((HtmlComponent)object).getHtml());
            fiche.append("<br>");
        }
    }
    return fiche.toString();
}
```

Classe ErpFicheHtml

La méthode setValuesAndWriteResponse()

Cette méthode est utilisée par la Servlet POST. Elle possède deux fonctions :

- setValues puisqu'il faut synchroniser les composants RAD avec les données des formulaires saisies.
- writeResponse puisqu'il faut écrire le JSON RAD de réponse ou retourner une nouvelle fiche créé.

Pour la partie setValues, l'objet webService est prévu à cet effet. Si on a utilisé la fonction RegisterRADEventExtra du script rad.js pour envoyer plus de données en complétant apiURL, alors il faut récupérer les données fournies dans le paramètre values(Map<String, String[]> values). Dans le cas contraire, comme ici, un simple appel à webService.setValues(values) suffit à transmettre toutes les données saisies dans le navigateur Web vers les composants RAD concernés.

Pour la partie writeResponses, puisque le JSON RAD est personnalisable, il faut le produire. Le webService se charge d'écrire sa partie avec l'appel à webService.writeResponses(out). Le reste du code consiste préparer la page qui contient les accolades du JSON. Cette exemple ci dessous rajoute la variable canclose qui sera récupérée via RegisterRADFillExtra .

```
public ErpFicheHtml setValuesAndWriteResponse(Map<String, String[]> values,
        HttpServletResponse response) throws IOException {
    webService.setValues(values);
    if (ficheSuivante == null) {
        response.setContentType("text/html");
        response.setCharacterEncoding("UTF-8");
        PrintWriter out = response.getWriter();
        out.print("{");
        webService.writeResponses(out);
        out.print(",");
        out.print("\"canclose\":\");
        if (canclose!=null) out.print(canclose);
        out.print("\");
        out.println("}");
        return null;
    }
    return (ErpFicheHtml)ficheSuivante;
}
```

Le reste de la classe

La méthode showMessage transmet la demande au webService et la méthode afficheFichier permet de créer une fiche de visualisation pour afficher un fichier PDF.

```
@Override
protected void showMessage(String message) {
    webService.addAlert(message);
}

@Override
protected void afficheFichier(String nomFichier) {...}
```

Améliorations

Améliorations visuelles.

Une amélioration possible du paquet RAD ou de l'application serait des améliorations visuelles. On pourrait changer les couleurs, rajouter des images. CSS permet de faire beaucoup sans toucher au reste du code. Il faut aussi retravailler le cadrage des zones numériques.

Améliorations de sécurité.

L'application est bien sécurisée rien que par le fait qu'il n'existe aucune URL directe vers une fonction spéciale de l'application. Il faut suivre le cheminement de l'application pour activer ses différentes fonctions.

Le manque de sécurité se situe au niveau de la session.

Si un utilisateur distrait consulte le client Dupont et que sur le même navigateur, il ouvre un second onglet sur l'application alors il verra la fiche Dupont. En restant sur ce même onglet, il passe au client suivant et affiche Durand. Il ferme ensuite l'onglet et retrouve le premier onglet resté sur l'affichage de Dupont. Alors à ce moment, si l'utilisateur clique sur modifier puis OK, il va écraser les données de Durant par les données de Dupont.

Pour résoudre ce risque, il faudra transmettre un compteur de génération de page et s'en servir pour constater que le client Web est bien en phase avec la situation réelle de l'application.

Améliorations ergonomiques.

C'est possible. Une amélioration que je considère comme prioritaire est l'ergonomie. Une bonne application peut se piloter entièrement au clavier. Ici, la gestion clavier est ignorée et laissée au système par défaut.

Plus de composants.

Il n'y en aura jamais assez.

package rad

Listeners :

```
public interface KeySelectListener {
    public void KeySelected(Long id);
}
public interface ListenerBooleanRAD {
    public boolean getBoolean(Object object);
}
public interface ListenerRAD {
    public void actionPerformed(RadComponent cpn);
}
```

DataRad :

```
public abstract class DataRAD {
    public static int maxFieldLength = 0;
    public DataRAD(Connection conn, String tableName, String keyField)
    public abstract RadStaticText createStaticText(String fieldName);
    public abstract RadTextField createTextField(String fieldName);
    public abstract RadRadioGroup createRadioGroup(String fieldName,
        String[] values, String[] labels);
    public abstract RadComboBox createSQLComboBox(String fieldName, String sql);
    public abstract RadButtonField createButtonField(String fieldName,
        String buttonlabel);
    public abstract DataRAD createLinkData(String linkField,
        String tableName, String keyField);
    public abstract RadTextField createDateField(String fieldName);
    public String getTableName()
    public DataFieldRAD createField(String fieldName)
    public void createDataField(String fieldName, DataFieldRAD dataField)
    public void createLink(String fieldName, DataRAD dr)
    public Long getKeyValue()
    public void setKeyValue(Long keyValue)
    public void readLink(String linkField, Long KeyValue)
    public void readLink(String linkField)
    public void setReadOnly(boolean readonly)
    public void clear()
    public boolean read(Long keyValue)
    public boolean update() throws SQLException
    public void create() throws SQLException
    public void doRefresh()
    public boolean doValidateSave()
    public DataRAD addRefresh(KeySelectListener ksl)
    public DataRAD addValidateSaveListener(ListenerBooleanRAD vl)
    protected int colIndex(String fieldName) throws SQLException
    protected int getFieldSize(int colIndex) throws SQLException
    public String getFieldValue(String fieldName)
    public Double getFieldValueDouble(String fieldName)
    public void setFieldValue(String fieldName, String value)
}
```


package rad

Composants visuels :

```
public interface RadComponent {
    void setVisible(boolean paramBoolean);
}

public interface RadButton extends RadComponent {
    public RadButton addActionListener(ListenerRAD al);
    public RadButton setConfirmation(String confirmation);
    public void setReadOnly(boolean readonly);
    public void setVisible(boolean visible);
    public void performAction();
}

public interface RadContainer extends RadComponent {
    public void addComponent(RadComponent cpn);
    public void addLabelComponent(String label, RadComponent radComponent) ;
}

public interface RadSQLGrid extends RadComponent {
    public RadSQLGrid addRowSelectListener(ListenerRAD al);
    public RadSQLGrid addDoubleClicListener(ListenerRAD al);
    public void refreshKey(long keyValue);
    public SQLGridSearch getGridSearch() ;
    public void setGridSearch(SQLGridSearch gs) ;
    public ArrayList<Long> getKeyValues() ;
    public int getSelectedLine() ;
    public long getSelectedKey() ;
    public void selectKey(long keyValue);
    public void filter() ;
    public void setOrderBy(String orderBy);
    public void setGroupBy(String groupBy);
}
```

Composants visuels liable à DataRad :

```
public interface DataFieldRAD {
    public void setValue(String value);
    public String getValue();
    public void setReadOnly(boolean readonly);
    public void setVisible(boolean visible);
}

public interface RadComboBox extends DataFieldRAD, RadComponent {
    public void addChangeListener(ListenerRAD al);
    public void addItem(String value, String label);
    public String getDisplayValue();
}

public interface RadButtonField extends DataFieldRAD, RadButton {}

public interface RadRadioGroup extends DataFieldRAD, RadComponent {
    public void addChangeListener(ListenerRAD al);
}

public interface RadStaticText extends DataFieldRAD, RadComponent {}

public interface RadTextField extends DataFieldRAD, RadComponent {
    void addChangeListener(ListenerRAD paramListenerRAD);
}
```

package rad

Autres :

```
public abstract class RadComponentFactory {
    public abstract RadTextField createTextField(String name);
    public abstract RadTextField createTextField(String name,int size);
    public abstract RadStaticText createStaticText(String name);
    public abstract RadButton createButton(String name,String text);
    public abstract RadContainer createContainer(String name);
    public abstract RadRadioGroup createRadRadioGroup(String name,
        String[] values, String[] labels);
    public abstract RadComboBox createSQLComboBox(String name,Connection conn,
        String sql);
    public abstract RadSQLGrid createRadSQLGrid(String name,Connection conn,
        String keyField, String sql, int hiddenFieldsCount,
        SQLGridCalculatedField[] calculatedFields) ;
    public static void FillSqlComboBox(RadComboBox cb, Connection conn, String sql)
}

public abstract class SQLGridCalculatedField {
    public int col;
    public String title;
    public SQLGridCalculatedField(int col, String title)
    public abstract String Calculate(ResultSet rs) throws SQLException;
}

public abstract class SQLGridSearch {
    public RadSQLGrid grille;
    public RadComponentFactory compFact;
    public SQLGridSearch(RadComponentFactory componentFactory)
    public abstract void init();
    public abstract String getFilter(ArrayList<String> values);
    public abstract RadContainer getContainer();
    public String getPadFieldName()
    public void search()
    public RadContainer createContainer(String nom)
    public RadTextField createTextField(String nomChamp)
    public RadRadioGroup createRadioGroup(String name, String[] values,
        String[] labels)
    public RadComboBox createSQLComboBox(String name, Connection conn, String sql)
    public SQLGridSearch addSearchListener(ListenerRAD al)
}
```

package rad.swing

Composants visuels Swing :

```
public abstract class SwingComponent implements RadComponent{
    public abstract Component getComponent();
}

public class SwingButton extends SwingComponent implements RadButton {
    public SwingButton(String text)
    public RadButton setConfirmation(String confirmation)
    @Override public void setVisible(boolean visible)
    @Override public RadButton addActionListener(ListenerRAD al)
    @Override public JButton getComponent()
    @Override public void setReadOnly(boolean readonly)
    @Override public void performAction()
}

public class SwingContainer extends SwingComponent implements RadContainer {
    public SwingContainer()
    @Override public void addComponent(RadComponent cpn)
    @Override public JPanel getComponent()
    @Override public void addLabelComponent(String label, RadComponent radComponent)
    @Override public void setVisible(boolean visible)
}

public class SwingSQLGrid extends SwingComponent implements RadSQLGrid {
    public SwingSQLGrid(Connection conn, String KeyField, String sql,
        int hiddenFieldsCount, SQLGridCalculatedField[] calculatedFields)
    @Override public void setVisible(boolean visible)
    @Override public Component getComponent()
    @Override public int getSelectedLine()
    @Override public long getSelectedKey()
    @Override public void filter()
    @Override public void selectKey(long keyValue)
    @Override public void refreshKey(long keyValue)
    @Override public RadSQLGrid addDoubleClicListener(ListenerRAD al)
    @Override public RadSQLGrid addRowSelectListener(ListenerRAD al)
    @Override public void setGridSearch(SQLGridSearch gs)
    @Override public SQLGridSearch getGridSearch()
    @Override public ArrayList<Long> getKeyValues()
    @Override public JScrollPane getScrollPane()
    @Override public Component getSearchPane()
    @Override public void setOrderBy(String orderBy)
    @Override public void setGroupBy(String groupBy)
}
```

package rad.swing

Composants visuels Swing liable à DataRad :

```
public class SwingButtonField extends SwingButton implements RadButtonField {
    public SwingButtonField(String text)
    @Override public void setValue(String value)
    @Override public String getValue()
}
```

```
public class SwingComboBox extends SwingComponent implements RadComboBox {
    public SwingComboBox()
    @Override public void addItem(String value, String label)
    @Override public void setValue(String value)
    @Override public String getValue()
    @Override public String getDisplayValue()
    @Override public void setReadOnly(boolean readonly)
    @Override public void setVisible(boolean visible)
    @Override public Component getComponent()
    @Override public void addChangeListener(ListenerRAD al)
}
```

```
public class SwingDateField extends SwingTextField {
    public SwingDateField()
    @Override public void setValue(String value)
    @Override public String getValue()
}
```

```
public class SwingRadioGroup extends SwingComponent implements RadRadioGroup {
    public SwingRadioGroup(String[] values, String[] labels)
    @Override public void setValue(String value)
    @Override public String getValue()
    @Override public void setReadOnly(boolean readonly)
    @Override public void setVisible(boolean visible)
    @Override public JPanel getComponent()
    @Override public void addChangeListener(ListenerRAD al)
}
```

```
public class SwingStaticText extends SwingComponent implements RadStaticText {
    public SwingStaticText()
    @Override public void setValue(String value)
    @Override public void setReadOnly(boolean readonly)
    @Override public String getValue()
    @Override public void setVisible(boolean visible)
    @Override public JLabel getComponent()
}
```

```
public class SwingTextField extends SwingComponent implements RadTextField {
    public SwingTextField(int size)
    @Override public void setValue(String value)
    @Override public void setReadOnly(boolean readonly)
    @Override public String getValue()
    @Override public void setVisible(boolean visible)
    @Override public void addChangeListener(ListenerRAD al)
    @Override public JPasswordField getComponent()
}
```

package rad.swing

DataRad :

```
public class DataSwing extends DataRAD {
    public DataSwing(Connection conn, String tableName, String keyField)
    @Override public JTextField createTextField(String fieldName)
    @Override public JTextField createDateField(String fieldName)
    @Override public SwingStaticText createStaticText(String fieldName)
    @Override public SwingRadioGroup createRadioGroup(String fieldName, String[] values,
        String[] labels)
    @Override public SwingComboBox createSQLComboBox(String fieldName, String sql)
    @Override public SwingButtonField createButtonField(String fieldName,
        String buttonlabel)
    @Override public DataSwing createLinkData(String linkField, String tableName,
        String keyField)
}
```

Autres Swing:

```
public class SwingComponentFactory extends RadComponentFactory {
    @Override public RadTextField createTextField(String name)
    @Override public RadTextField createTextField(String name, int size)
    @Override public RadContainer createContainer(String name)
    @Override public RadRadioGroup createRadRadioGroup(String name, String[] values,
        String[] labels)
    @Override public SwingComboBox createSQLComboBox(String name, Connection conn,
        String sql)
    @Override public RadSQLGrid createRadSQLGrid(String nom, Connection conn,
        String KeyField, String sql, int hiddenFieldsCount,
        SQLGridCalculatedField[] calculatedFields)
    @Override public RadButton createButton(String name, String text)
    @Override public RadStaticText createStaticText(String name)
}
```

package rad.html

Composants visuels Html :

```
public abstract class HtmlComponent implements RadComponent {
    public HtmlComponent(String name)
    public abstract String getHtml()
    public String getName()
    public void setName(String name)
    public String getClassName()
    public void setClassName(String className)
    public String getRadInit()
    public void setRadInit(String radInit)
    public WebService getWebService()
    public void setWebService(WebService webService)
    public void setValue(String value)
    public String getValue()
    public void setDefault()
    public void setReadOnly(boolean readonly)
    public boolean isReadOnly()
    public void setVisible(boolean visible)
    public String getRADEvent(String eventName)
    public void triggerEvent(String eventName)
    public String getResponseFunctions()
    public String getResponseProperties()
    public String getResponseDataValues()
    public String getResponseClasses()
    protected String HtmlReadOnly()
    protected String htmlInit()
    protected String htmlExtra()
    protected String htmlClass()
    protected String htmlServiceValue()
    protected String htmlServiceDataValue()
    static public String HTMLString(String value)
    static public String JSONString(String value)
}

public class HtmlButton extends HtmlComponent implements RadButton {
    public HtmlButton(String name, String text)
    public RadButton setConfirmation(String confirmation)
    public RadButton addActionListener(ListenerRAD al)
    @Override protected String HtmlReadOnly()
    @Override public void triggerEvent(String eventName)
    @Override public String getHtml()
    @Override public String getResponseProperties()
    @Override public void performAction()
}

public class HtmlContainer extends HtmlComponent implements RadContainer {
    public HtmlContainer(String name)
    @Override public void addComponent(RadComponent cpn)
    @Override public void addLabelComponent(String label, RadComponent radComponent)
    @Override public String getHtml()
    public void addHtml(String html)
}
```

package rad.html

Composants visuels Html (suite) :

```
public class HtmlSQLGrid extends HtmlComponent implements RadSQLGrid {
    public HtmlSQLGrid(String name, Connection conn,
        String KeyField, String sql, int hiddenFieldsCount,
        SQLGridCalculatedField[] calculatedFields)
    public ArrayList<Long> getKeyValues()
    public void setGridSearch(SQLGridSearch gs)
    public SQLGridSearch getGridSearch()
    public String getSearchHtml()
    public RadSQLGrid addRowSelectListener(ListenerRAD al)
@Override public String getHtml()
    public String getTableHtml()
    public void filter()
    public void refreshKey(long keyValue)
@Override public void setValue(String value)
    public void setSelectedLine(Integer value)
    public Long getLongValue()
@Override public void setReadOnly(boolean readonly)
@Override public void selectKey(long keyValue)
@Override public String getResponseProperties()
@Override public String getResponseFunctions()
@Override public void triggerEvent(String eventName)
    public RadSQLGrid addDoubleClickListener(ListenerRAD al)
@Override public int getSelectedLine()
@Override public long getSelectedKey()
@Override public void setDefault()
@Override public void setOrderBy(String orderBy)
@Override public void setGroupBy(String groupBy)
}
```

package rad.html

Composants visuels Html liable à DataRad :

```
public class HtmlButtonField extends HtmlButton implements RadButtonField {
    public HtmlButtonField(String name, String text)
    @Override public void setValue(String value)
    @Override public String getValue()
}

public class HtmlComboBox extends HtmlComponent implements RadComboBox {
    public HtmlComboBox(String name)
    @Override public void addChangeListener(ListenerRAD al)
    @Override public String HtmlReadOnly()
    @Override public String getHtml()
    @Override public void setValue(String value)
    @Override public String getValue()
    @Override public String getDisplayValue()
    @Override public void triggerEvent(String eventName)
    @Override public String getResponseProperties()
    public void addItem(String value, String label)
}

public class HtmlDateField extends HtmlTextField {
    public HtmlDateField(String name)
    @Override public String getHtml()
}

public class HtmlRadioGroup extends HtmlComponent implements RadRadioGroup {
    public HtmlRadioGroup(String nomChamp, String[] values, String[] labels)
    @Override public String getHtml()
    @Override public void setValue(String value)
    @Override public String getResponseClasses()
    @Override public String getValue()
    @Override public String getResponseProperties()
    @Override public void addChangeListener(ListenerRAD al)
    @Override public void triggerEvent(String eventName)
}

public class HtmlStaticText extends HtmlComponent implements RadStaticText {
    public HtmlStaticText(String nomChamp)
    @Override public String getHtml()
    @Override public void setValue(String value)
    @Override public String getValue()
    public int getIntegerValue()
    public long getLongValue()
    @Override public String getResponseProperties()
}
```


package rad.html

Composants visuels Html liable à DataRad (suite) :

```
public class HtmlTextField extends HtmlComponent implements RadTextField {
    protected ListenerRAD onChangeListener = null;
    public HtmlTextField(String name)
    public void setInputSize(int inputSize)
    public void setVisualSize(int visualSize)
    public String getVisualSize()
    public String getInputSize()
    @Override public String getHtml()
    @Override public void setValue(String value)
    @Override public String getValue()
    @Override public void triggerEvent(String eventName)
    public int getIntegerValue()
    public long getLongValue()
    @Override public String getResponseProperties()
    public void addChangeListener(ListenerRAD al)
}
```

package rad.html

DataRad :

```
public class DataHtml extends DataRAD {
    public DataHtml(WebService webService, Connection conn,
        String tableName, String keyField)
    @Override public HtmlTextField createTextField(String fieldName)
    @Override public HtmlTextField createDateField(String fieldName)
    @Override public HtmlStaticText createStaticText(String fieldName)
    @Override public HtmlRadioGroup createRadioGroup(String fieldName,
        String[] values, String[] labels)
    @Override public HtmlComboBox createSQLComboBox(String fieldName, String sql)
    @Override public DataHtml createLinkData(String linkField,
        String tableName, String keyField)
    @Override public HtmlButtonField createButtonField(String fieldName,
        String buttonlabel)
}
```

Autres Html :

```
public class HtmlComponentFactory extends RadComponentFactory {
    public HtmlComponentFactory(WebService webService)
    public WebService getWebService()
    @Override public RadTextField createTextField(String name)
    @Override public RadTextField createTextField(String name, int size)
    @Override public HtmlStaticText createStaticText(String name)
    @Override public HtmlContainer createContainer(String name)
    @Override public HtmlRadioGroup createRadRadioGroup(String name,
        String[] values, String[] labels)
    @Override public HtmlComboBox createSQLComboBox(String name, Connection conn,
        String sql)
    @Override public RadSQLGrid createRadSQLGrid(String nom, Connection conn,
        String keyField, String sql, int hiddenFieldsCount,
        SQLGridCalculatedField[] calculatedFields)
    @Override public RadButton createButton(String name, String text)
}

public class WebService {
    public WebService(String apiUrl, String name)
    public String getName()
    public void addAlert(String alert)
    public String getApiUrl()
    public void setApiUrl(String apiUrl)
    public void addComponent(HtmlComponent component)
    public void setDefault()
    public void setValues(Map<String, String[]> values)
    public void writeResponses(PrintWriter out) throws IOException
    public void writeResponseFunctions(PrintWriter out)
    public void writeResponseProperties(PrintWriter out)
    public void writeResponseClasses(PrintWriter out)
    public void writeResponseDataValues(PrintWriter out)
    public void writeResponseAlerts(PrintWriter out)
}
```

package rad.html

Composants visuels Html supplémentaires :

```
public class HtmlImage extends HtmlComponent {
    public HtmlImage(String name, String src)
    @Override public String getHtml()
    @Override public String getResponseProperties()
}
```

Composants visuels Html nécessitant la bibliothèque gfx.js:

```
public class HtmlCanvas extends HtmlComponent {
    public HtmlCanvas(String name, int width, int height)
    @Override public String getHtml()
    public void writeResponseDrawImage(PrintWriter out)
    @Override public String getResponseFunctions()
    public void selectDrawImage(HtmlImage image)
    public void drawImage(int imageX, int imageY,
                          int imageWidth, int imageHeight,
                          int DestinationX, int DestinationY,
                          int DestinationWidth, int DestinationHeight)
}

public class HtmlKeyboard extends HtmlComponent {
    public HtmlKeyboard(String name)
    @Override public String getHtml()
    @Override public void setValue(String value)
    public boolean hasNext()
    public int nextKey()
    public void setValues(Map<String, String[]> values)
}
```

Pensées philosophiques presque complètement hors sujet

1) « Code it right, code it once ».

Cette phrase se lit dans les deux sens.

Pour coder bien, il faut coder une seule fois.

Cela signifie qu'il faut se soucier de créer un code réutilisable. Il faut bien factoriser le code au maximum en découpant le code en sous-programmes. Chaque fois que l'on fait du copier-coller de code, c'est une erreur. Il faut à la place créer une bibliothèque de code réutilisable.

Pour coder bien, il faut coder une seule fois.

Cela signifie que lorsque l'on écrit un programme, il faut dès le début produire un code impeccable. Il faut que le code prévoie tous les cas possibles et que chaque nom de variable, procédure, fonction méthode, classe ou propriété soit précisément choisis. Il faut pouvoir valider le code et ne plus à avoir à le modifier. Face à un problème informatique, si on doit coder le programme, il existe dans ce cas quasiment une infinité de solutions possible. Mais il n'existe finalement presque qu'une seule solution idéale. Une solution où tout est optimisé, factorisé et utilisant une logique et des noms de variables parfaitement explicites.

2) Découvrir plutôt qu'inventer.

Voici une histoire drôle :

Un sculpteur expose sa sculpture et il est félicité pour la beauté de sa création. Comble de la modestie, le sculpteur rétorque que l'œuvre a toujours existé mais il n'a fait que la dégager de la roche. (ha ! ha !)

En sciences, il existe des découvertes, tels que les nombre premiers : Les nombres premiers sont immatériels. Ils existent. On ne peut que les découvrir, on ne peut pas les inventer.

Il existe aussi des inventions formidables qui sont parfois artistiques. Mais beaucoup peuvent être considéré comme des découvertes. La roue est une découverte, pas une invention. Une invention possède forcément des alternatives possible. On pourrait inventer une roue carré, ou hexagonale. Mais une roue est ronde, c'est le principe de la découverte.

En informatique, il existe des découvertes, par exemple tous les langages ont besoin de marquer un bloc de code par une balise de début et une balise de fin. En pascal on utilise les mot clé « begin » et « end ». En Java on utilise des accolades ouvrantes ou fermantes. Ce marquage de bloc est donc une découverte puisque c'est universel, mais le choix des marqueurs est inventé.

En conclusion, il existe des inventions et des découvertes, mais il faut plutôt chercher à découvrir plutôt qu'à inventer. La découverte est éternelle et immatérielle alors que l'invention est partielle et résulte de choix fait par son créateur.

Au début de l'algorithmie on invente une méthode de tri d'un tableau en permutant ses valeurs. Après plusieurs inventions, on invente finalement le principe du tri à bulle. Après analyse on extrait le code minimal et optimisé du tri à bulle. A ce niveau, le tri à bulle devient une découverte et ce n'est plus une invention. Cette forme d'algorithme sera valide pour l'éternité, tout comme les autres algorithmes de tri plus efficaces qui furent découverts après.