



Ressourcesinformatiques

 + QUIZ

Version en ligne

**OFFERTE !**

pendant 1 an

# Delphi 10.3

## Programmation Orientée Objet en environnement Windows

En téléchargement



code des exemples

Thierry GRASSIA





Les éléments à télécharger sont disponibles à l'adresse suivante :  
**<http://www.editions-eni.fr>**  
Saisissez la référence ENI de l'ouvrage **RIDELPH** dans la zone de recherche et validez. Cliquez sur le titre du livre puis sur le bouton de téléchargement.

## Chapitre 1

### Introduction

1. À qui s'adresse ce livre ?	19
2. Delphi	20
3. Un peu de vocabulaire	20
4. Installation	22
4.1 Téléchargement	22
4.2 Licence	23
4.3 Installation	24

## Chapitre 2

### Prise en main de l'IDE de Delphi 10.3

1. Présentation	27
2. Mise en pratique	32
2.1 Création d'un nouveau projet	32
2.2 Ajouts de quelques composants sur la fiche	36
2.3 Mise en forme	39
2.4 Définir un gestionnaire d'évènement	51
2.5 Menu Chercher	54
3. Le débogueur intégré de Delphi	56
3.1 À quoi sert-il ?	56
3.2 Comment ?	57
3.3 Utilisation du débogueur	57

4. Refactoring . . . . .	63
4.1 Renommer une classe . . . . .	64
4.2 Extraire une méthode d'un bloc de code . . . . .	65
5. Renommage de nom de fichier . . . . .	66
6. Options du projet . . . . .	67
7. Conclusion . . . . .	70

## Chapitre 3

### Delphi, le langage Pascal objet

1. Introduction . . . . .	71
2. Structure des fichiers Delphi . . . . .	71
2.1 Création d'un projet d'exemple . . . . .	72
2.2 Le fichier .pas . . . . .	75
2.3 Le fichier .dpr . . . . .	78
2.4 Le fichier .dproj . . . . .	81
3. Le mot réservé uses . . . . .	81
4. Les variables et les constantes . . . . .	82
4.1 Déclaration . . . . .	82
4.2 Portée . . . . .	83
5. Les types de variables Delphi . . . . .	85
5.1 Les types entiers . . . . .	85
5.2 Les types décimaux . . . . .	86
5.3 Les types booléens . . . . .	86
5.4 Les types caractères . . . . .	87
5.5 Les types chaînes de caractères . . . . .	87
5.6 Les types énumération . . . . .	90
5.7 Les tableaux . . . . .	91
5.7.1 Tableau statique . . . . .	91
5.7.2 Tableau dynamique . . . . .	92
5.7.3 Manipulations de base . . . . .	93
5.8 Les variants . . . . .	94

5.9	Le record	95
5.9.1	Définition	95
5.9.2	Le mot réservé with	96
6.	Les opérateurs	97
6.1	Opérateur d'affectation	97
6.2	Opérations mathématiques	97
6.3	Opérations logiques booléennes	102
6.4	Comparaisons numériques	102
7.	Procédures et fonctions	105
7.1	Procédure	107
7.2	Fonction	108
7.3	Passage de paramètres	110
8.	Instructions basiques	115
8.1	Les tests de conditions	115
8.2	If then else	116
8.3	Case of else	117
8.4	Les instructions d'itération	118
9.	Les directives de compilation conditionnelles	124
10.	Les fonctions de conversion	126
10.1	Les fonctions de conversion pour les chaînes de caractères	126
10.2	Les fonctions de conversion pour les nombres	127
10.3	Les fonctions de conversion récemment ajoutées	127
11.	Les pointeurs	129
11.1	Définition	130
11.2	Utilisation	130
11.3	Implémentation	131
11.4	Conclusion	132

## Chapitre 4 Concept objet en Delphi

1. Introduction à la programmation orientée objet . . . . .	133
2. Principes de la programmation objet. . . . .	134
2.1 Les champs . . . . .	134
2.2 Les méthodes. . . . .	134
2.3 Encapsulation de données . . . . .	135
2.4 Représentation UML et nommage . . . . .	135
2.5 Héritage. . . . .	136
2.6 Polymorphisme . . . . .	138
3. Définition de classe . . . . .	141
3.1 Constructeur. . . . .	143
3.2 Destructeur. . . . .	145
3.3 Le pointeur nil. . . . .	146
3.4 Pointeur interne Self. . . . .	146
3.5 Propriétés. . . . .	147
3.6 Propriété tableau. . . . .	150
3.7 Objets et pointeurs. . . . .	151
3.8 Évènements. . . . .	152
3.9 L'opérateur is. . . . .	157
3.10 L'opérateur as . . . . .	157
3.11 Champs et méthodes statiques . . . . .	159
4. Mise en pratique - Application pour la location de véhicules . . . . .	160
4.1 Convention de nommage. . . . .	161
4.2 Implémentation . . . . .	162
4.3 Utilisation. . . . .	170
4.4 Remarques. . . . .	173
5. Conclusion . . . . .	174

**Chapitre 5****Run Time Library, fondation de Delphi**

1. Les bibliothèques fournies par Delphi .....	175
2. RTL mise en pratique .....	176
2.1 La classe TObject .....	177
2.2 Les cas d'exception .....	178
2.3 La classe TPersistent et le système RTTI .....	184
2.4 Mise en pratique de l'introspection RTTI .....	185
2.5 La classe helper .....	189
2.6 Les listes en Delphi .....	192
2.6.1 TList .....	192
2.6.2 Exemple d'utilisation de TList .....	193
2.6.3 TObjectList .....	201
2.6.4 TArray .....	201
2.6.5 Les files (FIFO) TQueue et TObjectQueue .....	203
2.6.6 Les piles (LIFO) TStack et TObjectStack .....	206
2.6.7 Les listes de string TStringList .....	206
2.6.8 Les dictionnaires TDictionnary et TObjectDictionnary .....	217
2.7 Le système de fichiers TFile, TPath, TDirectory .....	219
2.7.1 TPath : Manipulation de chemins sur le disque .....	220
2.7.2 TDirectory : Parcours et manipulation de répertoire ..	222
2.7.3 TFile : Manipulation de fichier .....	224
2.8 Les flux, TStream et ses descendants .....	227
2.9 Opérations mathématiques .....	233
2.10 La manipulation de dates avec TDate et TTime .....	235
2.11 La base des registres TRegistry .....	240
2.12 Les expressions régulières avec TRegEx .....	244
2.13 Compression de fichier ou de répertoires .....	245
3. Les méthodes anonymes .....	246
3.1 Définition .....	246
3.2 Déclaration .....	247

3.3	Implémentation	247
3.4	Le principe de closure	248
3.5	Conclusion	250
4.	Les interfaces	250
4.1	Définition	250
4.2	Déclaration	251
4.3	Utilisation	252
4.4	Héritage d'interface	253
4.5	Utilisation de plusieurs interfaces, héritage multiple	254
4.6	Utilisation de même nom de méthode dans deux interfaces différentes	255
4.7	Pour finir	255
5.	Conclusion	256

## Chapitre 6

### Les applications Windows avec la VCL

1.	La VCL : Visual Component Library	257
1.1	Introduction	257
1.2	Les classes de base	258
1.2.1	La classe TComponent	258
1.2.2	La classe TControl	259
1.2.3	La classe TWinControl	263
2.	La classe TApplication	264
3.	Les applications Windows VCL	265
3.1	Une application Windows VCL simple	266
3.2	Une application MDI	269
4.	Création d'une IHM avec Delphi	272
4.1	Paramètres d'affichage de TForm	272
4.1.1	Méthodes relatives à l'apparition d'une fenêtre	273
4.1.2	Évènements de TForm	274

4.2	La palette de composants standards	274
4.2.1	Le bouton : TButton	275
4.2.2	Zone de texte : TLabel	276
4.2.3	Case à cocher : TCheckBox	276
4.2.4	Choix multiples : TRadioButton	277
4.2.5	Choix multiple dans une liste : TComboBox	278
4.2.6	Les composants d'édition de textes	279
4.2.7	Les menus	280
4.2.8	Les alignements avec les TPanel et les TGroupBox	284
4.3	Modifier la position d'un composant	289
4.4	Travailler avec des frames et réutiliser le contexte graphique	293
4.5	Sélectionner des fichiers (TOpenDialog) et des répertoires (TOpenFileDialog)	298
4.6	Présentation de données	299
4.6.1	Avec TPageControl	299
4.6.2	En tableau avec la TStringGrid	302
4.6.3	Présentation de données en arbre avec TTreeView	305
4.7	Zone avec barre de défilement : TScrollBar	309
4.8	Exécuter une action périodique avec TTimer	312
4.9	Applications VCL multifenêtres	313
4.10	Échanger des données entre deux fenêtres	315
4.10.1	Méthode par un appel modal	315
4.10.2	Méthode par évènement	316
4.11	Récupérer des évènements du clavier	318
4.12	Faire un glisser-déposer (Drag&Drop)	320
4.13	Afficher des images	322
4.13.1	Le contrôle TImage	322
4.13.2	Le contrôle TImageList	323
4.13.3	Le contrôle TImageCollection	324
4.14	Travailler avec les styles prédéfinis en VCL	327
4.15	Créer et utiliser son propre style VCL	329

4.16 Configurer une application . . . . .	339
4.16.1 Par ligne de commande . . . . .	339
4.16.2 Par fichier de configuration. . . . .	340
5. Conclusion . . . . .	341

## Chapitre 7

### Les bibliothèques (dll) et les packages (bpl)

1. Définitions . . . . .	343
2. Mise en pratique de l'utilisation d'une dll . . . . .	345
2.1 Création . . . . .	345
2.2 Export de routines . . . . .	346
2.3 Exploitation d'une dll dans un exécutable . . . . .	347
2.3.1 Chargement statique de la dll. . . . .	349
2.3.2 Chargement dynamique de la dll . . . . .	351
2.3.3 La directive delayed . . . . .	353
2.4 Passage de chaînes de caractères . . . . .	353
2.4.1 La méthode ShareMem . . . . .	353
2.4.2 La méthode par type primitif (pchar integer) . . . . .	356
2.4.3 Évènement levé dans une dll et utilisé dans le programme hôte . . . . .	358
3. Les packages BPL Delphi . . . . .	363
3.1 Création d'un package runtime . . . . .	363
3.2 Création d'un package design . . . . .	372
4. Conclusion . . . . .	380

**Chapitre 8****Représentation de données : formats XML et JSON**

1. Introduction . . . . .	381
2. Format XML . . . . .	382
2.1 Écriture du fichier XML, sérialisation directe . . . . .	382
2.2 Lecture du fichier XML, désérialisation directe . . . . .	393
3. XML Data Binding . . . . .	398
3.1 Utilisation du moteur XML Data Binding . . . . .	399
3.2 Sérialisation . . . . .	406
3.3 Désérialisation . . . . .	407
4. Format JSON . . . . .	408
4.1 Présentation . . . . .	408
4.2 Syntaxe . . . . .	409
4.3 Sérialisation JSON . . . . .	410
4.3.1 Par une structure objet définie manuellement . . . . .	410
4.3.2 Par réflexion . . . . .	413
4.4 Désérialisation . . . . .	415
4.4.1 Par la classe TJSONObject . . . . .	415
4.4.2 Par réflexion . . . . .	418
5. Conclusion . . . . .	419

**Chapitre 9****Traitement de tâches asynchrones**

1. Description d'un processus . . . . .	421
2. Les threads dans un processus . . . . .	422
3. Pourquoi utiliser du multithreading ? . . . . .	423
3.1 Effectuer plusieurs traitements en parallèle . . . . .	423
3.2 Donner une meilleure expérience utilisateur . . . . .	424
4. La classe TThread . . . . .	426
5. Thread transitoire TThread . . . . .	428

6. Thread transitoire par procédure anonyme (TThread.CreateAnonymousThread) . . . . .	431
7. Thread persistant . . . . .	432
8. Les concepts de ressource partagée et de section critique . . . . .	436
9. Une liste thread safe, l'interface d'échange privilegiée entre deux threads . . . . .	439
9.1 Par TCriticalSection . . . . .	439
9.2 Par TThreadList et TMonitor . . . . .	442
10. PPL : la bibliothèque de programmation parallèle . . . . .	443
10.1 Pourquoi cette bibliothèque en plus de TThread ? . . . . .	444
10.2 Thread transitoire par TTask.Start . . . . .	444
11. Synchronisation de deux threads grâce à l'appel Future . . . . .	445
12. Exécuter N actions en parallèle avec TParallel . . . . .	449
12.1 Présentation . . . . .	449
12.2 Exécution de la même action en parallèle TParallel.For. . . . .	449
12.3 Exécution de N méthodes différentes avec TParallel.Join . . . . .	454
12.4 Remarques. . . . .	457
13. Recommandations . . . . .	457
14. Conclusion . . . . .	458

## Chapitre 10

### Création d'un service Windows

1. Définition . . . . .	459
2. Configuration . . . . .	460
2.1 Configuration du type d'exécution . . . . .	460
2.2 Configuration du compte associé . . . . .	460
2.3 Configuration des dépendances. . . . .	461
3. Implémentation . . . . .	461
3.1 Configuration du service . . . . .	463
3.2 Implémentation des évènements . . . . .	464

- 3.3 Exécution et débogage ..... 468
  - 3.3.1 L'évènement OnExecute ..... 471
  - 3.3.2 Utilisation d'un thread secondaire ..... 476
- 4. Débogage de service Windows ..... 477
  - 4.1 Par fichier de log ..... 477
  - 4.2 S'attacher au processus par l'IDE ..... 483
- 5. Conclusion ..... 485

**Chapitre 11**  
**Communication interprocessus (IPC)**

- 1. Introduction ..... 487
- 2. Communication via message Windows ..... 488
  - 2.1 Récupération du handle de fenêtre destinataire ..... 490
  - 2.2 Envoi du message ..... 492
  - 2.3 Réception du message ..... 494
  - 2.4 À savoir ..... 494
- 3. Communication via TCP/IP ..... 495
  - 3.1 Définitions ..... 495
  - 3.2 Implémentation d'un système de communication client/serveur ..... 497
    - 3.2.1 Implémentation du serveur ..... 498
    - 3.2.2 Implémentation du client ..... 500
    - 3.2.3 Gestion du contenu du message ..... 502
    - 3.2.4 Écriture d'un protocole ..... 504
    - 3.2.5 Envoi d'un message ..... 506
    - 3.2.6 Réception d'un message ..... 508
    - 3.2.7 Intégration dans le client ..... 513
    - 3.2.8 Intégration dans le serveur ..... 514
    - 3.2.9 Identification de l'émetteur du message ..... 515

3.3	Implémentation d'un tchat . . . . .	520
3.3.1	Modifications du client de tchat . . . . .	526
3.3.2	Modifications de l'interface graphique . . . . .	529
3.3.3	Rafraîchissement des onglets . . . . .	531
3.3.4	Ajout d'un message entrant . . . . .	533
3.3.5	Envoi d'un message . . . . .	534
3.3.6	Autres modifications du serveur de tchat . . . . .	535
3.4	Utilisation du tchat dans un réseau d'ordinateurs . . . . .	540
3.4.1	Modélisation Model View Controller . . . . .	542
3.4.2	Création d'un service . . . . .	543
4.	Conclusion . . . . .	545

## Chapitre 12

### Connectivité aux bases de données

1.	Définitions . . . . .	547
2.	Les différents frameworks . . . . .	551
2.1	Le framework BDE . . . . .	551
2.2	Le framework dbGo . . . . .	551
2.3	Le framework dbExpress . . . . .	552
2.4	Le framework FireDAC . . . . .	552
3.	Le SQL en bref - fonctions de base . . . . .	553
4.	Les composants FireDAC . . . . .	555
5.	Une application de e-commerce avec back-office . . . . .	557
5.1	Design de la base de données . . . . .	557
5.2	Étapes de développement . . . . .	559
5.3	Installation MariaDB . . . . .	560
6.	L'application de back-office . . . . .	563
6.1	Configuration des composants . . . . .	566
6.1.1	Configuration de TFDCConnection : FDConnection . . . . .	566
6.1.2	Configuration du TFDTTable : FDTTableProduct . . . . .	568
6.1.3	Configuration du TDataSource : DataSourceProduct . . . . .	568

- 6.1.4 Configuration de TDBGrid : DBGridProduct . . . . . 569
- 6.1.5 Configuration du TDBNavigator :  
DBNavigatorProduct . . . . . 571
- 6.1.6 Configuration du TDBImage : DBImageProduct . . . . . 572
- 6.2 Interfaçages . . . . . 572
  - 6.2.1 Interfaçage du champ description . . . . . 572
  - 6.2.2 Interfaçage du champ category . . . . . 573
  - 6.2.3 Interfaçage du champ image . . . . . 574
- 6.3 Personnalisation du rendu de la grille . . . . . 576
  - 6.3.1 Insertion de l'image dans la colonne image . . . . . 576
  - 6.3.2 Modification des couleurs de texte  
ou de police dans la grille . . . . . 578
  - 6.3.3 Modification de la hauteur des lignes de la grille . . . . . 580
- 6.4 Tri de la grille par valeur d'un champ . . . . . 581
- 7. Conclusion . . . . . 582

**Chapitre 13**

**Utilisation du design pattern MVC**

- 1. Introduction . . . . . 583
- 2. Design pattern MVC . . . . . 583
  - 2.1 Le modèle . . . . . 584
  - 2.2 La vue . . . . . 585
  - 2.3 Le contrôleur . . . . . 585
  - 2.4 Les avantages . . . . . 585
- 3. Développement de l'application . . . . . 586
  - 3.1 Fonctionnalités . . . . . 586
  - 3.2 Couches MVC . . . . . 586
    - 3.2.1 Les vues . . . . . 586
    - 3.2.2 Les modèles . . . . . 587
    - 3.2.3 Les contrôleurs . . . . . 587
  - 3.3 Navigation entre les pages . . . . . 592
  - 3.4 Normaliser les échanges entre MVC . . . . . 596

3.5	Les vues	599
3.5.1	La vue « Login »	600
3.5.2	La vue "Choix d'objets"	611
3.5.3	La vue gestion de compte	626
3.5.4	La vue « Validation de panier »	644
3.6	Création d'une facture	651
3.6.1	Préparation et création des éléments nécessaires à l'opération	651
3.6.2	Création d'un modèle	652
3.6.3	Récupération des données	655
3.6.4	Design du rapport	659
3.6.5	Génération du fichier	660
3.7	Calcul du montant total de la commande par procédure stockée	662
4.	Conclusion	666

### Chapitre 14 Attributs, annotations et RTTI

1.	Introduction	667
2.	Principe du mapping propriété/colonne	668
3.	Attributs Delphi	669
3.1	Généralités	669
3.2	Modélisation	670
4.	Un moteur d'accès aux données générique	677
4.1	Implémentation	677
4.2	Le cas des propriétés objets	692
4.2.1	Sauvegarde dans la base de données	692
4.2.2	Chargement dans la base de données	694
4.3	Gestion de listes	695
4.4	Les tables de jointure, relation OneToMany	698
5.	Conclusion	711

## Chapitre 15 FireDAC, FireMonkey et LiveBinding

1. FireMonkey . . . . .	713
2. LiveBinding . . . . .	714
3. Migration des outils back-office et E-Commerce en FireMonkey . .	715
3.1 Le composant TListView . . . . .	715
3.2 Liaison visuelle avec une source de données FireDAC : LiveBinding . . . . .	721
4. Interagir avec la TListView . . . . .	724
4.1 Le clic . . . . .	724
4.2 Le double clic . . . . .	725
5. Visualisation Maître/Détail . . . . .	725
5.1 Présentation . . . . .	725
5.2 Maître/Détail utilisateur/commandes . . . . .	726
5.3 Maître/Détail commande/objets achetés . . . . .	732
5.4 Relation Maître/Détail, commande/adresse de livraison . . . .	736
6. Modification de la valeur issue de la source de donnée . . . . .	739
6.1 Cas d'application . . . . .	739
6.2 La propriété CustomFormat . . . . .	740
6.3 Les routines de conversion . . . . .	742
6.4 Créer ses propres routines de conversion . . . . .	746
7. Conclusion . . . . .	747

## Chapitre 16 Serveur d'application REST

1. Définition : application distribuée . . . . .	749
2. Protocole HTTP . . . . .	750
3. Exemple d'échanges de données : information de météo . . . . .	751
3.1 Par le protocole HTTP . . . . .	751
3.2 Par le protocole SOAP (Simple Object Access Protocol) . . . .	752

3.3	Par une architecture REST	754
4.	Service web et web-méthodes REST	756
4.1	Cas général	756
4.2	Cas de l'authentification	758
4.2.1	La problématique	758
4.2.2	Les méthodes d'authentification HTTP	758
4.2.3	Les différentes options	759
4.2.4	Implémentation choisie	762
5.	Implémentation de l'application côté serveur	778
5.1	Le projet Web Server Application	779
5.2	Mise en place du serveur Apache	782
5.2.1	Installation d'Apache	783
5.2.2	Configuration du serveur Apache	783
5.2.3	Intégration du module	785
5.3	Édition des points d'entrée	786
5.4	Utilisation du REST Debugger de Delphi	788
5.5	Passage de paramètres du client vers le serveur	791
5.6	Implémentation de l'enregistrement d'un utilisateur	791
5.7	La classe TJsonResult	796
5.8	Le cas de l'authentification : le login	799
5.9	Récupération des produits	805
5.10	Mettre à jour un utilisateur	809
5.11	Passer une commande	813
5.12	Gérer la montée en charge avec l'utilisation d'un pool de connexion	819
6.	Conclusion	824

**Chapitre 17**

**Client REST multi-plateforme FireMonkey**

- 1. Introduction ..... 825
  - 1.1 Design graphique ..... 825
  - 1.2 Échanges entre le client et le serveur ..... 829
  - 1.3 Implémentation ..... 831
- 2. Appel REST simple, enregistrement d'un utilisateur ..... 836
- 3. Authentification REST de type Basic et login d'un utilisateur ... 843
- 4. Utilisation du Livebinding avec un modèle de données ..... 846
  - 4.1 Utilisation d'un TDataGeneratorAdapter  
pour définir un prototype d'échange ..... 848
  - 4.2 Encapsulation dans un TAdapterBindSource ..... 849
  - 4.3 Liaison avec un TListView ..... 851
  - 4.4 Gestion de panier ..... 854
    - 4.4.1 Design graphique ..... 854
    - 4.4.2 Implémentation ..... 856
  - 4.5 Passer une commande ..... 859
  - 4.6 Gestion compte utilisateur ..... 863
- 5. Conclusion ..... 870

**Chapitre 18**

**Travailler avec les styles FireMonkey**

- 1. Présentation ..... 871
- 2. Modifier ou créer un style de composant ..... 872
  - 2.1 Le Style Designer ..... 872
  - 2.2 Modification du style TLabel ..... 875
  - 2.3 Créer un nouveau style ..... 878

3.	Créer et utiliser un style spécifique pour une TListBox. . . . .	881
3.1	Création d'un style TListBox. . . . .	881
3.2	Ajout d'éléments stylisés dans la TListBox. . . . .	883
3.2.1	Exemple : Affectation du prix. . . . .	884
3.2.2	Exemple : Affectation de l'image . . . . .	884
4.	Intégration dans le client REST. . . . .	886
5.	Conclusion . . . . .	891
	Conclusion . . . . .	893
	Index . . . . .	895

## Chapitre 4

# Concept objet en Delphi

### 1. Introduction à la programmation orientée objet

Il s'agit d'une approche différente de la programmation procédurale. Si un programme informatique écrit de manière procédurale est la réponse à la question "que veut-on faire?" le même programme écrit de manière orientée objet répond à la question "de quoi parle-t-on?". On découpe fonctionnellement le programme à réaliser par rôles, par concept ou par entités physiques modélisées dans le programme. Les entités ainsi définies sont appelées "objet".

Nous allons étudier dans ce chapitre l'exemple d'une application de gestion de location de véhicules. Rapidement, on peut énumérer quelques rôles et entités :

- Le véhicule (quoi ?)
- Les utilisateurs de l'application (client ou administrateur) (qui ?)
- L'agence de location (où ?)
- La réservation de location (date de début/date de fin) (quand ?)

En général, on modélise les objets à travers le langage UML qui permet de décrire schématiquement la structure, le comportement de chaque objet. On décrit aussi en UML le comportement des différents objets les uns par rapport aux autres. Au final, on peut décrire l'intégralité d'un programme informatique en UML.

Au niveau de l'implémentation en Delphi, les objets sont l'extension des records vus dans le chapitre précédent (pour rappel, le record est un type qui permet d'agréger tout un jeu d'autres types). On dit qu'ils possèdent une structure interne décrite par les champs et un comportement décrit par les méthodes que la classe implémente.

En Delphi, les objets sont décrits par la définition de la classe à laquelle il appartient.

## 2. Principes de la programmation objet

### 2.1 Les champs

Un champ est une variable interne à un objet. Comme toute variable, ce champ possède un type (entier, chaîne de caractères ou même une autre classe) et est manipulé par les méthodes de la classe à laquelle il appartient.

Un champ peut être :

- un champ d'instance, qui est propre à chaque objet.
- un champ statique ou champ de classe qui est propre à toute la classe. La valeur de cette variable est la même pour toutes les instances de la classe.

### 2.2 Les méthodes

Les méthodes sont les routines qui permettent de manipuler les champs de la classe. En fonction d'un certain état d'entrée A, selon la méthode exécutée M, l'objet se retrouve dans un autre état B.

L'ensemble formé par les champs et les méthodes est appelé "membres de la classe".

Une méthode peut être :

- une méthode d'instance, n'agissant que sur un seul objet (instance de la classe) à la fois,
- une méthode statique ou méthode de classe qui n'agit pas sur des variables d'instance mais uniquement sur des variables de classe. Elle est indépendante de toute instance de la classe (objet).

## 2.3 Encapsulation de données

L'intérêt d'un objet réside dans le fait qu'il rassemble toutes les données (champs) dont il a besoin au sein d'une même classe ainsi que les moyens de les gérer (méthodes). Dans ce principe d'encapsulation, il existe également une notion de visibilité qui permet de masquer ou non certaines données de l'objet que l'on manipule de l'extérieur.

Cela permet de garantir une cohérence des données qui ne pourront être modifiées de l'extérieur qu'à travers les méthodes visibles.

Dans le principe d'encapsulation, il existe deux modes de visibilité :

- Privé
- Public

Un membre privé est inaccessible vu de l'extérieur à la classe concernée.

## 2.4 Représentation UML et nommage

Voici un exemple de représentation UML (*Unified Modeling Language*) :

Ci-après est définie la classe **TExemple**. En général, on définit le nom d'une classe Delphi avec un T majuscule devant pour signifier « type ». Bien qu'un type et une classe sont deux notions différentes, il faut garder en mémoire que Delphi est une extension du langage Pascal où il n'existait que des types ; il s'agit uniquement d'une continuité de notation.

TExemple
-FAttributePrivate
+AttributePublic
+MethodPublic()
-MethodPrivate()

La représentation de cette classe possède deux zones : celle du dessus pour les champs et l'autre en dessous pour les méthodes. Ainsi, dans la seconde zone, après chaque nom sont ajoutés les symboles **()** spécifiant un appel de routine.

#### ■ Remarque

*En Delphi, quand une routine ne possède pas de paramètre d'entrée, les symboles **()** ne sont pas obligatoires. Cependant, il est préférable de les rajouter pour une meilleure visibilité de code.*

Nous trouvons donc deux champs : **FAttributePrivate** et **AttributePublic**.

Le **-** devant **FAttributePrivate** signifie que la visibilité est privée et le **+** devant **AttributePublic** signifie que la visibilité est publique. Pour un champ privé, on met en général un **F** pour désigner «field» (champ en anglais).

De même, pour les méthodes, on trouve **MethodPublic()** en visibilité publique et **MethodPrivate()** en visibilité privée.

## 2.5 Héritage

Il s'agit du mécanisme qui permet de créer une classe B à partir d'une classe existante A. On dit qu'on a « spécialisé » ou « enrichi » la classe A par ce mécanisme.

La notion de visibilité évoquée précédemment rentre en compte également :

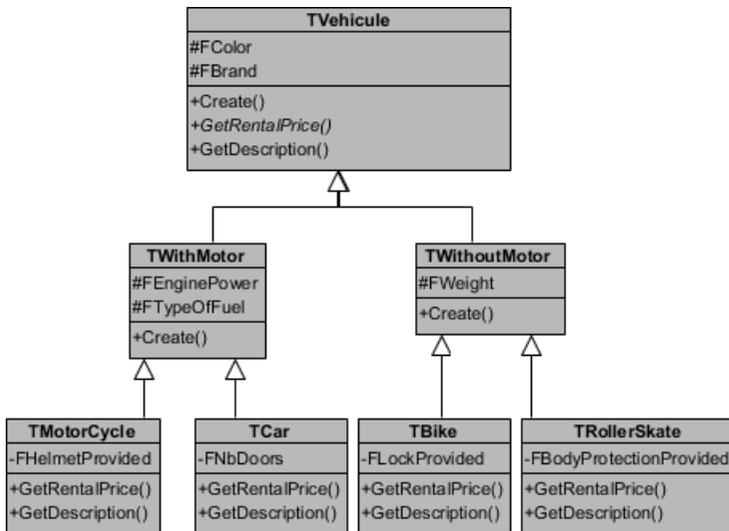
La classe B aura connaissance par le mécanisme d'héritage des membres publics ou protégés de la classe A.

Le terme « protégé » est une visibilité à mi-chemin entre public et privé, car comme un membre privé, un membre protégé ne peut pas être manipulé de l'extérieur de la classe mais il est accessible dans la classe héritée.

En UML, le symbole utilisé pour noter une visibilité protégée est le **#**.

Pour reprendre l'exemple de l'application de location de véhicules qui permet de gérer des vélos, des rollers des voitures et des motos, il est possible de définir deux catégories de véhicules : une avec moteur et une autre sans moteur.

Ainsi, on peut imaginer une hiérarchie de classe avec comme classe de base **TVehicle** avec deux classes héritées, une pour les véhicules à moteur (**TWithMotor**) et l'autre sans moteur (**TWithoutMotor**). À ce stade les classes **TWithMotor** et **TWithoutMotor** représentent encore deux concepts très généraux. Il est alors possible d'ajouter un héritage supplémentaire pour modéliser l'ensemble des produits disponibles à la location, à savoir les vélos (**TBike**), les rollers (**TRollerSkate**), les voitures (**TCar**) et les motos (**TMotorCycle**) qui représentent spécifiquement des objets du monde réel.



Dans un schéma UML, l'héritage entre deux classes est modélisé par une flèche sur fond blanc. Dans le schéma ci-dessus, on a bien **TBike** qui hérite de **TWithoutMotor** et **TWithoutMotor** qui hérite à son tour de **TVehicle**.

Pour rappel, les champs précédés d'un # sont de visibilité protégée.

En reprenant l'exemple ci-dessus, **TCar** et **TMotorCycle** auront accès à la propriété **FTypeOfFuel** qui représente le type de carburant de l'objet à louer. Cette notion n'apparaît pas dans la hiérarchie des véhicules sans moteur **TBike** ou **TRollerSkate** car elle est non applicable pour ces objets du monde réel. Un vélo ne possède pas de type de carburant.

Ainsi, on considère que tous les véhicules possèdent une couleur (**FColor**) et une marque (**FBrand**).

Pour un véhicule sans moteur, le poids devient important, et c'est en positionnant **FWeight** dans la classe **TWithoutMotor** que l'on met l'accent sur cette notion qui sera partagée entre les vélos et les rollers.

Pour un véhicule avec un moteur, il est important de connaître sa puissance et le type de carburant. Ainsi, on positionne ces deux champs dans **TWithMotor** (**FEnginePower** et **FTypeOfFuel**).

Dans les classes modélisant les produits à louer, on retrouve en visibilité privée le casque fourni pour une moto (**FHelmetProvided** dans **TMotorCycle**), le nombre de portes pour une voiture (**FNbDoors** dans **TCar**), le cadenas fourni pour un vélo (**FLockProvided** dans **TBike**) et la protection corporelle genoux, coudes et tête fournie pour un roller (**FBodyProtectionProvided** dans **TRoller**).

Ainsi, on a bien une voiture qui possède un nombre de portes, un type de carburant, une puissance, une marque et une couleur. De même, on a bien un vélo qui possède un cadenas, un poids, une couleur et une marque.

## 2.6 Polymorphisme

Par polymorphisme on entend la possibilité qu'à une classe enfant de redéfinir une méthode et donc un comportement de la classe parent.

Une méthode qui porte le même nom entre le parent et l'enfant peut posséder la même signature et une implémentation différente.

Pour rappel, la signature d'une routine est constituée de son type de retour (rien, s'il s'agit d'une procédure ou du type renvoyé par une fonction) associé à la liste ordonnée des paramètres d'entrée de la routine. La signature de procédure **MyProc(param1:string;param2:integer);** est différente de **procédure MyProc(param1,param2:string);**. Dans un cas, les paramètres d'entrée sont un entier et une chaîne de caractères et dans l'autre, deux chaînes de caractères.

Grâce à ce principe de polymorphisme, le compilateur effectue une liaison dynamique des appels. Ainsi, soit la méthode enfant ou la méthode parent est appelée en fonction du contexte d'exécution.

Il ne faut pas confondre polymorphisme et surcharge car dans le cas du polymorphisme les paramètres d'entrée de la méthode sont les mêmes et dans le cas de la surcharge ils sont différents.

La surcharge est le processus de multiples définitions de routines avec des paramètres différents abordés dans le chapitre précédent. Ces définitions de méthodes utilisent la directive **overload**.

En Delphi, pour utiliser le polymorphisme à travers l'héritage, il faut utiliser au minimum les directives **virtual** et **override** et éventuellement la directive **abstract**.

- La directive **virtual** indique que la méthode est candidate à une redéfinition dans une classe héritée.
- La directive **override** indique que c'est la méthode de la classe héritée qui doit être exécutée.
- La directive **abstract** indique que cette méthode n'a pas d'implémentation dans la classe courante. Il faut utiliser conjointement **abstract** avec **virtual** car sinon cette méthode ne sera jamais implémentée dans aucun contexte, hérité ou non. Dans le cas où une méthode notée comme **abstract** ne possède aucune implémentation dans la hiérarchie de classe, une erreur d'appel à une méthode non implémentée est possible. Un avertissement est levé par le compilateur lors de la compilation.