

projet_global

Généré par Doxygen 1.8.18

1 Index hiérarchique	1
1.1 Hiérarchie des classes	1
2 Index des classes	3
2.1 Liste des classes	3
3 Index des fichiers	5
3.1 Liste des fichiers	5
4 Documentation des classes	7
4.1 Référence de la classe Actif	7
4.1.1 Description détaillée	8
4.1.2 Documentation des constructeurs et destructeur	8
4.1.2.1 Actif() [1/2]	8
4.1.2.2 ~Actif()	8
4.1.2.3 Actif() [2/2]	8
4.1.3 Documentation des fonctions membres	8
4.1.3.1 getInstance()	8
4.1.3.2 getLetter()	9
4.1.3.3 getPoste()	9
4.1.3.4 operator=()	9
4.1.4 Documentation des données membres	10
4.1.4.1 instance	10
4.2 Référence de la classe CompteHierarchie	10
4.2.1 Description détaillée	11
4.2.2 Documentation des constructeurs et destructeur	11
4.2.2.1 CompteHierarchie() [1/2]	11
4.2.2.2 ~CompteHierarchie()	12
4.2.2.3 CompteHierarchie() [2/2]	12
4.2.3 Documentation des fonctions membres	13
4.2.3.1 accepte()	13
4.2.3.2 ajouterFils()	13
4.2.3.3 getDernierMemento()	14
4.2.3.4 getFils()	14
4.2.3.5 getId()	14
4.2.3.6 getMementos()	14
4.2.3.7 getNom()	15
4.2.3.8 getPoste()	15
4.2.3.9 getPremierMemento()	15
4.2.3.10 getType()	16
4.2.3.11 operator=()	16
4.2.3.12 saveState()	16
4.2.3.13 supprimerCompte()	17

4.2.3.14 supprimerFils()	17
4.2.3.15 toString()	17
4.2.3.16 undo()	18
4.2.4 Documentation des fonctions amies et associées	18
4.2.4.1 CompteManager	18
4.2.4.2 HierarchieBuilder	18
4.2.4.3 VisiteurFree	18
4.2.5 Documentation des données membres	18
4.2.5.1 fils	19
4.2.5.2 id	19
4.2.5.3 nom	19
4.2.5.4 poste	19
4.3 Référence de la classe CompteManager	19
4.3.1 Description détaillée	21
4.3.2 Documentation des constructeurs et destructeur	21
4.3.2.1 CompteManager() [1/2]	21
4.3.2.2 ~CompteManager()	21
4.3.2.3 CompteManager() [2/2]	21
4.3.3 Documentation des fonctions membres	21
4.3.3.1 affiche()	22
4.3.3.2 ajouterFils()	22
4.3.3.3 bilan()	22
4.3.3.4 charger()	23
4.3.3.5 cloturer()	23
4.3.3.6 crediterCompte()	23
4.3.3.7 debiterCompte()	23
4.3.3.8 deplacerFils()	24
4.3.3.9 getCompte()	24
4.3.3.10 getInstance()	25
4.3.3.11 getOriginator()	25
4.3.3.12 getPere()	25
4.3.3.13 getSolde()	26
4.3.3.14 operator=()	26
4.3.3.15 rapprocherCompte()	26
4.3.3.16 releve()	27
4.3.3.17 resultat()	27
4.3.3.18 sauver()	27
4.3.3.19 setPath()	28
4.3.3.20 supprimerCompte()	28
4.3.4 Documentation des données membres	28
4.3.4.1 instance	28
4.3.4.2 o	29

4.3.4.3 path	29
4.3.4.4 racine	29
4.4 Référence de la classe CompteReel	29
4.4.1 Description détaillée	30
4.4.2 Documentation des constructeurs et destructeur	30
4.4.2.1 CompteReel() [1/2]	30
4.4.2.2 ~CompteReel()	30
4.4.2.3 CompteReel() [2/2]	30
4.4.3 Documentation des fonctions membres	31
4.4.3.1 accepte()	31
4.4.3.2 ajouterFils()	31
4.4.3.3 getDernierMemento()	31
4.4.3.4 getMementos()	33
4.4.3.5 getPremierMemento()	33
4.4.3.6 getType()	33
4.4.3.7 operator=()	33
4.4.3.8 saveState()	34
4.4.3.9 supprimerCompte()	34
4.4.3.10 supprimerFils()	34
4.4.3.11 toString()	35
4.4.3.12 undo()	35
4.4.4 Documentation des fonctions amies et associées	35
4.4.4.1 CompteManager	36
4.4.4.2 HierarchieBuilder	36
4.4.4.3 VisiteurFree	36
4.4.5 Documentation des données membres	36
4.4.5.1 mementos	36
4.5 Référence de la classe CompteVirtuel	36
4.5.1 Description détaillée	37
4.5.2 Documentation des constructeurs et destructeur	37
4.5.2.1 CompteVirtuel() [1/2]	38
4.5.2.2 ~CompteVirtuel()	38
4.5.2.3 CompteVirtuel() [2/2]	38
4.5.3 Documentation des fonctions membres	38
4.5.3.1 accepte()	38
4.5.3.2 ajouterFils()	39
4.5.3.3 getDernierMemento()	39
4.5.3.4 getMementos()	39
4.5.3.5 getPremierMemento()	40
4.5.3.6 getType()	40
4.5.3.7 operator=()	40
4.5.3.8 saveState()	40

4.5.3.9 supprimerCompte()	41
4.5.3.10 supprimerFils()	41
4.5.3.11 toString()	42
4.5.3.12 undo()	42
4.5.4 Documentation des fonctions amies et associées	42
4.5.4.1 CompteManager	42
4.5.4.2 HierarchieBuilder	42
4.5.4.3 VisiteurFree	43
4.6 Référence de la classe Depense	43
4.6.1 Description détaillée	43
4.6.2 Documentation des constructeurs et destructeur	44
4.6.2.1 Depense() [1/2]	44
4.6.2.2 ~Depense()	44
4.6.2.3 Depense() [2/2]	44
4.6.3 Documentation des fonctions membres	44
4.6.3.1 getInstance()	44
4.6.3.2 getLetter()	45
4.6.3.3 getPoste()	45
4.6.3.4 operator=()	45
4.6.4 Documentation des données membres	45
4.6.4.1 instance	45
4.7 Référence de la classe Exception	46
4.7.1 Description détaillée	46
4.7.2 Documentation des constructeurs et destructeur	46
4.7.2.1 Exception()	46
4.7.3 Documentation des fonctions membres	47
4.7.3.1 getCode()	47
4.7.3.2 getMessage()	47
4.7.4 Documentation des données membres	47
4.7.4.1 code_erreur	47
4.7.4.2 message	47
4.8 Référence de la classe ExceptionComptabilite	48
4.8.1 Description détaillée	48
4.8.2 Documentation des énumérations membres	48
4.8.2.1 CodeExcepComptabilite	48
4.8.3 Documentation des constructeurs et destructeur	49
4.8.3.1 ExceptionComptabilite()	49
4.9 Référence de la classe ExceptionFichier	49
4.9.1 Description détaillée	49
4.9.2 Documentation des énumérations membres	49
4.9.2.1 CodeExcepFichier	49
4.9.3 Documentation des constructeurs et destructeur	50

4.9.3.1 ExceptionFichier()	50
4.10 Référence de la classe ExceptionHierarchie	50
4.10.1 Description détaillée	51
4.10.2 Documentation des énumérations membres	51
4.10.2.1 CodeExcepHierarchie	51
4.10.3 Documentation des constructeurs et destructeur	51
4.10.3.1 ExceptionHierarchie()	51
4.11 Référence de la classe ExceptionTransaction	51
4.11.1 Description détaillée	52
4.11.2 Documentation des énumérations membres	52
4.11.2.1 CodeExcepTransaction	52
4.11.3 Documentation des constructeurs et destructeur	52
4.11.3.1 ExceptionTransaction()	52
4.12 Référence de la classe HierarchieBuilder	53
4.12.1 Description détaillée	53
4.12.2 Documentation des fonctions membres	53
4.12.2.1 creerAvecFichier()	53
4.12.2.2 creerEmpty()	54
4.12.2.3 creerFils()	54
4.12.3 Documentation des fonctions amies et associées	55
4.12.3.1 CompteManager	55
4.12.4 Documentation des données membres	55
4.12.4.1 idcompte	55
4.13 Référence de la classe Memento	55
4.13.1 Description détaillée	56
4.13.2 Documentation des constructeurs et destructeur	56
4.13.2.1 Memento()	56
4.13.3 Documentation des fonctions membres	56
4.13.3.1 getDate()	56
4.13.3.2 getSolde()	56
4.13.4 Documentation des données membres	56
4.13.4.1 date	57
4.13.4.2 solde	57
4.14 Référence de la classe Operation	57
4.14.1 Description détaillée	58
4.14.2 Documentation des constructeurs et destructeur	58
4.14.2.1 ~Operation()	58
4.14.2.2 Operation()	58
4.14.3 Documentation des fonctions membres	59
4.14.3.1 getCredit()	59
4.14.3.2 getDebit()	59
4.14.3.3 getIdCompte()	59

4.14.3.4 isCompte()	59
4.14.3.5 operator<()	60
4.14.3.6 operator<=()	60
4.14.3.7 operator>()	60
4.14.3.8 operator>=()	60
4.14.4 Documentation des fonctions amies et associées	60
4.14.4.1 operator<<	61
4.14.5 Documentation des données membres	61
4.14.5.1 compte	61
4.14.5.2 credit	61
4.14.5.3 debit	61
4.14.5.4 transaction	61
4.14.5.5 TransactionBuilder	61
4.15 Référence de la classe Originator	62
4.15.1 Description détaillée	62
4.15.2 Documentation des constructeurs et destructeur	62
4.15.2.1 Originator()	62
4.15.3 Documentation des fonctions membres	62
4.15.3.1 getDate()	63
4.15.3.2 getSolde()	63
4.15.3.3 restoreMemento()	63
4.15.3.4 saveState()	63
4.15.3.5 setSolde()	63
4.15.4 Documentation des données membres	64
4.15.4.1 date	64
4.15.4.2 solde	64
4.16 Référence de la classe Passif	64
4.16.1 Description détaillée	65
4.16.2 Documentation des constructeurs et destructeur	65
4.16.2.1 Passif() [1/2]	65
4.16.2.2 ~Passif()	65
4.16.2.3 Passif() [2/2]	65
4.16.3 Documentation des fonctions membres	66
4.16.3.1 getInstance()	66
4.16.3.2 getLetter()	66
4.16.3.3 getPoste()	66
4.16.3.4 operator=()	66
4.16.4 Documentation des données membres	67
4.16.4.1 instance	67
4.17 Référence de la classe PosteAD	67
4.17.1 Description détaillée	67
4.17.2 Documentation des fonctions membres	67

4.17.2.1 crediter()	67
4.17.2.2 debiter()	68
4.18 Référence de la classe PosteCompte	68
4.18.1 Description détaillée	68
4.18.2 Documentation des fonctions membres	69
4.18.2.1 crediter()	69
4.18.2.2 debiter()	69
4.18.2.3 getLetter()	69
4.18.2.4 getPoste()	69
4.19 Référence de la classe PostePR	70
4.19.1 Description détaillée	70
4.19.2 Documentation des fonctions membres	70
4.19.2.1 crediter()	70
4.19.2.2 debiter()	71
4.20 Référence de la classe Racine	71
4.20.1 Description détaillée	72
4.20.2 Documentation des constructeurs et destructeur	72
4.20.2.1 Racine() [1/2]	72
4.20.2.2 ~Racine()	72
4.20.2.3 Racine() [2/2]	72
4.20.3 Documentation des fonctions membres	72
4.20.3.1 crediter()	72
4.20.3.2 debiter()	73
4.20.3.3 getInstance()	73
4.20.3.4 getLetter()	73
4.20.3.5 getPoste()	74
4.20.3.6 operator=()	74
4.20.4 Documentation des données membres	74
4.20.4.1 instance	74
4.21 Référence de la classe Recette	74
4.21.1 Description détaillée	75
4.21.2 Documentation des constructeurs et destructeur	75
4.21.2.1 Recette() [1/2]	75
4.21.2.2 ~Recette()	75
4.21.2.3 Recette() [2/2]	75
4.21.3 Documentation des fonctions membres	76
4.21.3.1 getInstance()	76
4.21.3.2 getLetter()	76
4.21.3.3 getPoste()	76
4.21.3.4 operator=()	76
4.21.4 Documentation des données membres	77
4.21.4.1 instance	77

4.22 Référence de la classe Transaction	77
4.22.1 Description détaillée	78
4.22.2 Documentation des constructeurs et destructeur	78
4.22.2.1 Transaction() [1/2]	78
4.22.2.2 Transaction() [2/2]	78
4.22.2.3 ~Transaction()	79
4.22.3 Documentation des fonctions membres	79
4.22.3.1 getDate()	79
4.22.3.2 getListeOperations()	79
4.22.3.3 getMemo()	80
4.22.3.4 getReference()	80
4.22.3.5 getStringDate()	80
4.22.3.6 getValide()	80
4.22.3.7 operator<()	81
4.22.3.8 operator<=()	81
4.22.3.9 operator>()	81
4.22.3.10 operator>=()	81
4.22.3.11 setValide()	81
4.22.4 Documentation des données membres	82
4.22.4.1 CompteManager	82
4.22.4.2 date	82
4.22.4.3 memo	82
4.22.4.4 operations	82
4.22.4.5 reference_transaction	82
4.22.4.6 TransactionBuilder	82
4.22.4.7 valide	83
4.23 Référence de la classe TransactionBuilder	83
4.23.1 Description détaillée	83
4.23.2 Documentation des constructeurs et destructeur	84
4.23.2.1 TransactionBuilder()	84
4.23.2.2 ~TransactionBuilder()	84
4.23.3 Documentation des fonctions membres	84
4.23.3.1 creerOperation()	84
4.23.3.2 creerTransaction()	85
4.23.3.3 creerTransactionAvecFichier()	85
4.23.3.4 detruireTransaction()	86
4.23.4 Documentation des données membres	86
4.23.4.1 instance_builder	86
4.23.4.2 TransactionManager	86
4.24 Référence de la classe TransactionManager	87
4.24.1 Description détaillée	88
4.24.2 Documentation des constructeurs et destructeur	88

4.24.2.1	TransactionManager()	88
4.24.2.2	~TransactionManager()	88
4.24.3	Documentation des fonctions membres	88
4.24.3.1	ajouterOperation()	88
4.24.3.2	ajouterTransaction()	89
4.24.3.3	charger()	89
4.24.3.4	editerTransaction()	90
4.24.3.5	getInstance()	91
4.24.3.6	getListeTransactions()	91
4.24.3.7	getListeTransactionsParCompte()	91
4.24.3.8	getListeTransactionsParValide()	91
4.24.3.9	getTransaction()	92
4.24.3.10	sauver()	92
4.24.3.11	supprimerTransaction()	93
4.24.4	Documentation des données membres	93
4.24.4.1	instance	93
4.24.4.2	path	93
4.24.4.3	transactions	94
4.25	Référence de la classe Visiteur	94
4.25.1	Description détaillée	94
4.25.2	Documentation des fonctions membres	94
4.25.2.1	visiter() [1/2]	94
4.25.2.2	visiter() [2/2]	95
4.26	Référence de la classe VisiteurAffichage	95
4.26.1	Description détaillée	96
4.26.2	Documentation des constructeurs et destructeur	96
4.26.2.1	VisiteurAffichage() [1/2]	96
4.26.2.2	VisiteurAffichage() [2/2]	96
4.26.3	Documentation des fonctions membres	96
4.26.3.1	afficherLigne()	96
4.26.3.2	operator=()	97
4.26.3.3	visiter() [1/2]	97
4.26.3.4	visiter() [2/2]	97
4.26.4	Documentation des données membres	98
4.26.4.1	CompteManager	98
4.26.4.2	indentation	98
4.27	Référence de la classe VisiteurFree	98
4.27.1	Description détaillée	99
4.27.2	Documentation des constructeurs et destructeur	99
4.27.2.1	VisiteurFree() [1/2]	99
4.27.2.2	VisiteurFree() [2/2]	99
4.27.3	Documentation des fonctions membres	99

4.27.3.1 operator=()	100
4.27.3.2 visiter() [1/2]	100
4.27.3.3 visiter() [2/2]	100
4.27.4 Documentation des données membres	100
4.27.4.1 CompteManager	101
4.28 Référence de la classe VisiteurGetSolde	101
4.28.1 Description détaillée	101
4.28.2 Documentation des constructeurs et destructeur	101
4.28.2.1 VisiteurGetSolde() [1/2]	102
4.28.2.2 VisiteurGetSolde() [2/2]	102
4.28.3 Documentation des fonctions membres	102
4.28.3.1 getSolde()	102
4.28.3.2 operator=()	102
4.28.3.3 visiter() [1/2]	103
4.28.3.4 visiter() [2/2]	103
4.28.4 Documentation des données membres	103
4.28.4.1 CompteManager	103
4.28.4.2 solde	104
4.29 Référence de la classe VisiteurPere	104
4.29.1 Description détaillée	104
4.29.2 Documentation des constructeurs et destructeur	104
4.29.2.1 VisiteurPere() [1/2]	105
4.29.2.2 VisiteurPere() [2/2]	105
4.29.3 Documentation des fonctions membres	105
4.29.3.1 getResultat()	105
4.29.3.2 operator=()	105
4.29.3.3 visiter() [1/2]	106
4.29.3.4 visiter() [2/2]	106
4.29.4 Documentation des données membres	106
4.29.4.1 CompteManager	106
4.29.4.2 recherche	106
4.29.4.3 resultat	107
4.30 Référence de la classe VisiteurPoste	107
4.30.1 Description détaillée	107
4.30.2 Documentation des constructeurs et destructeur	107
4.30.2.1 VisiteurPoste() [1/2]	108
4.30.2.2 VisiteurPoste() [2/2]	108
4.30.3 Documentation des fonctions membres	108
4.30.3.1 changePoste()	108
4.30.3.2 getResultat()	108
4.30.3.3 operator=()	108
4.30.3.4 visiter() [1/2]	109

4.30.3.5 visiter() [2/2]	109
4.30.4 Documentation des données membres	109
4.30.4.1 CompteManager	109
4.30.4.2 p	110
4.30.4.3 resultat	110
4.31 Référence de la classe VisiteurRecherche	110
4.31.1 Description détaillée	111
4.31.2 Documentation des constructeurs et destructeur	111
4.31.2.1 VisiteurRecherche() [1/2]	111
4.31.2.2 VisiteurRecherche() [2/2]	111
4.31.3 Documentation des fonctions membres	111
4.31.3.1 getResultat()	111
4.31.3.2 operator=()	111
4.31.3.3 visiter() [1/2]	112
4.31.3.4 visiter() [2/2]	112
4.31.4 Documentation des données membres	112
4.31.4.1 CompteManager	112
4.31.4.2 idcompte	113
4.31.4.3 resultat	113
5 Documentation des fichiers	115
5.1 Référence du fichier Actif.cpp	115
5.2 Référence du fichier Actif.h	115
5.3 Référence du fichier CompteHierarchie.cpp	115
5.4 Référence du fichier CompteHierarchie.h	115
5.4.1 Documentation du type de l'énumération	116
5.4.1.1 Type	116
5.5 Référence du fichier CompteManager.cpp	116
5.6 Référence du fichier CompteManager.h	116
5.7 Référence du fichier CompteReel.cpp	116
5.8 Référence du fichier CompteReel.h	117
5.9 Référence du fichier CompteVirtuel.cpp	117
5.10 Référence du fichier CompteVirtuel.h	117
5.11 Référence du fichier declarations.h	117
5.12 Référence du fichier Depense.cpp	117
5.13 Référence du fichier Depense.h	118
5.14 Référence du fichier Exception.h	118
5.15 Référence du fichier ExceptionCompte.h	118
5.16 Référence du fichier exceptiontransaction.h	118
5.17 Référence du fichier HierarchieBuilder.cpp	119
5.18 Référence du fichier HierarchieBuilder.h	119
5.19 Référence du fichier main.cpp	119

5.19.1 Documentation des macros	119
5.19.1.1 BUFF_COMPTE	119
5.19.1.2 BUFF_CREDIT	120
5.19.1.3 BUFF_DATE	120
5.19.1.4 BUFF_DEBIT	120
5.19.1.5 BUFF_INTITULE	120
5.19.1.6 BUFF_REF	120
5.19.2 Documentation des fonctions	120
5.19.2.1 afficherLigneTransaction()	120
5.19.2.2 buff()	120
5.19.2.3 main()	121
5.20 Référence du fichier Memento.cpp	121
5.21 Référence du fichier Memento.h	121
5.22 Référence du fichier operation.cpp	121
5.22.1 Documentation des fonctions	121
5.22.1.1 operator<<()	121
5.23 Référence du fichier operation.h	121
5.24 Référence du fichier Originator.h	122
5.25 Référence du fichier Passif.cpp	122
5.26 Référence du fichier Passif.h	122
5.27 Référence du fichier PosteAD.h	122
5.28 Référence du fichier PosteCompte.cpp	123
5.29 Référence du fichier PosteCompte.h	123
5.29.1 Documentation du type de l'énumération	123
5.29.1.1 Poste	123
5.30 Référence du fichier PostePR.h	123
5.31 Référence du fichier Racine.cpp	124
5.32 Référence du fichier Racine.h	124
5.33 Référence du fichier Recette.cpp	124
5.34 Référence du fichier Recette.h	124
5.35 Référence du fichier transaction.cpp	124
5.36 Référence du fichier transaction.h	125
5.37 Référence du fichier transactionbuilder.cpp	125
5.38 Référence du fichier transactionbuilder.h	125
5.39 Référence du fichier transactionmanager.cpp	125
5.40 Référence du fichier transactionmanager.h	125
5.41 Référence du fichier Visiteur.cpp	126
5.42 Référence du fichier Visiteur.h	126
5.43 Référence du fichier VisiteurAffichage.cpp	126
5.44 Référence du fichier VisiteurAffichage.h	126
5.45 Référence du fichier VisiteurFree.cpp	126
5.46 Référence du fichier VisiteurFree.h	126

5.47 Référence du fichier <code>VisiteurGetSolde.cpp</code>	127
5.48 Référence du fichier <code>VisiteurGetSolde.h</code>	127
5.49 Référence du fichier <code>VisiteurPere.cpp</code>	127
5.50 Référence du fichier <code>VisiteurPere.h</code>	127
5.51 Référence du fichier <code>VisiteurPoste.cpp</code>	127
5.52 Référence du fichier <code>VisiteurPoste.h</code>	128
5.53 Référence du fichier <code>VisiteurRecherche.cpp</code>	128
5.54 Référence du fichier <code>VisiteurRecherche.h</code>	128
Index	129

Chapitre 1

Index hiérarchique

1.1 Hiérarchie des classes

Cette liste d'héritage est classée approximativement par ordre alphabétique :

CompteHierarchie	10
CompteReel	29
CompteVirtuel	36
CompteManager	19
Exception	46
ExceptionComptabilite	48
ExceptionFichier	49
ExceptionHierarchie	50
ExceptionTransaction	51
HierarchieBuilder	53
Memento	55
Operation	57
Originator	62
PosteCompte	68
PosteAD	67
Actif	7
Depense	43
PostePR	70
Passif	64
Recette	74
Racine	71
Transaction	77
TransactionBuilder	83
TransactionManager	87
Visiteur	94
VisiteurAffichage	95
VisiteurFree	98
VisiteurGetSolde	101
VisiteurPere	104
VisiteurPoste	107
VisiteurRecherche	110

Chapitre 2

Index des classes

2.1 Liste des classes

Liste des classes, structures, unions et interfaces avec une brève description :

Actif		
	Actif permet de représenter les propriétés des comptes de poste Actif	7
CompteHierarchie		
	CompteHierarchie est la classe mère pour les éléments de la hiérarchie des comptes qui permet de manipuler tous les comptes via la même interface	10
CompteManager		
	CompteManager gère l'ensemble des Comptes	19
CompteReel		
	CompteReel représente les compte réels (les comptes sur lesquels auront lieu les opérations) .	29
CompteVirtuel		
	CompteVirtuel représente les Compte Virtuels (comptes destinés à regrouper des sous-compte sans enregistrer d'opération)	36
Depense		
	Depense permet de représenter les propriétés des comptes de poste Depense	43
Exception		
	Exception permet de gérer une erreur ainsi que son message	46
ExceptionComptabilite		
	ExceptionComptabilite est créée lors d'une opération de comptabilité illégale	48
ExceptionFichier		
	Traite les exceptions de la partie fichier	49
ExceptionHierarchie		
	ExceptionHierarchie est créée lors d'une opération illégale sur la hiérarchie de comptes	50
ExceptionTransaction		
	Traite les exceptions de la partie transaction	51
HierarchieBuilder		
	HierarchieBuilder gère la création de la hiérarchie des comptes	53
Memento		
	Le Memento permet de sauvegarder la date et le solde du dernier rapprochement	55
Operation		
	Objet représentant une opération comptable	57
Originator		
	Originator permet de manipuler les mementos	62
Passif		
	Passif permet de représenter les propriétés des comptes de poste Passif	64

PosteAD	
PosteAD est la classe mère des postes de compte : Actif et Depense . Elle a pour but de définir les méthodes créditer et débiter qui sont communes à ces deux classes	67
PosteCompte	
PosteCompte est la classe mère des différents postes de compte : Actif et Dépense , Passif et Recette , et la Racine	68
PostePR	
PostePR est la classe mère des postes de compte : Recette et Passif . Elle a pour but de définir les méthodes créditer et débiter qui sont communes à ces deux classes	70
Racine	
Racine permet de représenter le comportement d'un compte Racine , c'est-à-dire celui qui contient tous les autres comptes	71
Recette	
Recette permet de représenter le comportement des comptes de poste Recette	74
Transaction	
La transaction représente un ensemble d'opérations équilibrées effectuées à une même date entre plusieurs comptes	77
TransactionBuilder	
Le transaction builder permet de construire les transactions. Il utilise le pattern builder et singleton	83
TransactionManager	
Le transaction manager permet de gérer les transactions. Elle est basée sur les pattern Manager et Singleton	87
Visiteur	
Classe mères de différents visiteurs. Les visiteurs permettent de parcourir la hiérarchie des comptes	94
VisiteurAffichage	
VisiteurAffichage est un visiteur qui permet d'afficher la hiérarchie des comptes	95
VisiteurFree	
VisiteurFree est un visiteur qui permet de libérer la mémoire dans des comptes	98
VisiteurGetSolde	
VisiteurGetSolde est un visiteur qui permet de libérer la mémoire dans des comptes	101
VisiteurPere	
VisiteurPere est un visiteur qui permet de trouver le père d'un compte	104
VisiteurPoste	
VisiteurPoste est un visiteur qui permet de récupérer une liste de tous les comptes réels d'un poste	107
VisiteurRecherche	
VisiteurRecherche est un visiteur qui permet de rechercher un compte dans la hiérarchie des comptes	110

Chapitre 3

Index des fichiers

3.1 Liste des fichiers

Liste de tous les fichiers avec une brève description :

Actif.cpp	115
Actif.h	115
CompteHierarchie.cpp	115
CompteHierarchie.h	115
CompteManager.cpp	116
CompteManager.h	116
CompteReel.cpp	116
CompteReel.h	117
CompteVirtuel.cpp	117
CompteVirtuel.h	117
declarations.h	117
Depense.cpp	117
Depense.h	118
Exception.h	118
ExceptionCompte.h	118
exceptiontransaction.h	118
HierarchieBuilder.cpp	119
HierarchieBuilder.h	119
main.cpp	119
Memento.cpp	121
Memento.h	121
operation.cpp	121
operation.h	121
Originator.h	122
Passif.cpp	122
Passif.h	122
PosteAD.h	122
PosteCompte.cpp	123
PosteCompte.h	123
PostePR.h	123
Racine.cpp	124
Racine.h	124
Recette.cpp	124
Recette.h	124
transaction.cpp	124

transaction.h	125
transactionbuilder.cpp	125
transactionbuilder.h	125
transactionmanager.cpp	125
transactionmanager.h	125
Visiteur.cpp	126
Visiteur.h	126
VisiteurAffichage.cpp	126
VisiteurAffichage.h	126
VisiteurFree.cpp	126
VisiteurFree.h	126
VisiteurGetSolde.cpp	127
VisiteurGetSolde.h	127
VisiteurPere.cpp	127
VisiteurPere.h	127
VisiteurPoste.cpp	127
VisiteurPoste.h	128
VisiteurRecherche.cpp	128
VisiteurRecherche.h	128

Chapitre 4

Documentation des classes

4.1 Référence de la classe Actif

[Actif](#) permet de représenter les propriétés des comptes de poste [Actif](#).

```
#include <Actif.h>
```

Est dérivée de [PosteAD](#).

Fonctions membres publiques

- [Poste](#) [getPoste](#) () const noexcept override
Redéfinition de la méthode virtuelle [getPoste](#) de [PosteCompte](#). Cette méthode retourne le nom du poste actif : [ACTIF](#).
- string [getLetter](#) () const noexcept override
Redéfinition de la méthode virtuelle [getLetter](#) de [PosteCompte](#). Cette méthode renvoie la première lettre du nom d'un poste actif : [A](#).

Fonctions membres publiques statiques

- static const [PosteCompte](#) & [getInstance](#) () noexcept
Méthode qui permet d'obtenir l'instance [Actif](#).

Fonctions membres privées

- [Actif](#) ()=default
Constructeur privé.
- [~Actif](#) ()=default
Destructeur privé.
- [Actif](#) (const [Actif](#) &_a)=default
Constructeur par recopie privé.
- [Actif](#) & [operator=](#) (const [Actif](#) &_a)=default
Surcharge de l'opérateur d'affectation privé.

Attributs privés statiques

- static [Actif](#) [instance](#)
instance [Actif](#)

4.1.1 Description détaillée

[Actif](#) permet de représenter les propriétés des comptes de poste [Actif](#).

4.1.2 Documentation des constructeurs et destructeur

4.1.2.1 `Actif()` [1/2]

```
Actif::Actif ( ) [private], [default]
```

Constructeur privé.

4.1.2.2 `~Actif()`

```
Actif::~~Actif ( ) [private], [default]
```

Destructeur privé.

4.1.2.3 `Actif()` [2/2]

```
Actif::Actif (
    const Actif & _a ) [private], [default]
```

Constructeur par recopie privé.

Paramètres

↔	: const Actif &, référence constante sur le poste actif à recopier.
↔	
<u>↔</u>	
<i>a</i>	

4.1.3 Documentation des fonctions membres

4.1.3.1 `getInstance()`

```
static const PosteCompte & Actif::getInstance ( ) [inline], [static], [noexcept]
```

Méthode qui permet d'obtenir l'instance [Actif](#).

Renvoie

instance

4.1.3.2 getLetter()

```
string Actif::getLetter ( ) const [inline], [override], [virtual], [noexcept]
```

Redéfinition de la méthode virtuelle getLetter de [PosteCompte](#). Cette méthode renvoie la première lettre du nom d'un poste actif : A.

Renvoie

A

Implémente [PosteCompte](#).

4.1.3.3 getPoste()

```
Poste Actif::getPoste ( ) const [inline], [override], [virtual], [noexcept]
```

Redéfinition de la méthode virtuelle getPoste de [PosteCompte](#). Cette méthode retourne le nom du poste actif : ACTIF.

Renvoie

ACTIF

Implémente [PosteCompte](#).

4.1.3.4 operator=()

```
Actif & Actif::operator= (
    const Actif & _a ) [private], [default]
```

Surcharge de l'opérateur d'affectation privé.

Paramètres

↔	: const Actif &, référence constante sur le poste actif à affecter.
↔	
<i>a</i>	

4.1.4 Documentation des données membres

4.1.4.1 instance

`Actif` `Actif::instance` `[static]`, `[private]`

instance `Actif`

La documentation de cette classe a été générée à partir du fichier suivant :

- `Actif.h`
- `Actif.cpp`

4.2 Référence de la classe CompteHierarchie

`CompteHierarchie` est la classe mère pour les éléments de la hiérarchie des comptes qui permet de manipuler tous les comptes via la même interface.

```
#include <CompteHierarchie.h>
```

Dérivée par `CompteReel`, et `CompteVirtuel`.

Fonctions membres publiques

- virtual string `toString` () const noexcept=0
Méthode virtuelle qui retournera des informations sur les comptes.
- virtual void `accepte` (`Visiteur` &_v) noexcept=0
Méthode pour les visiteurs.
- string `getNom` () const noexcept
Méthode qui permet d'obtenir le nom du compte.
- `Poste` `getPoste` () const noexcept
Méthode qui retourne le poste d'un compte.
- virtual `Type` `getType` () const noexcept=0
Méthode qui retourne le type d'un Compte.
- const set< `CompteHierarchie` * > `getFils` () const noexcept
Méthode qui permet d'obtenir les fils d'un compte.
- virtual void `ajouterFils` (`CompteHierarchie` *_c)=0
Méthode qui sera redéfinie. Elle permettra d'ajouter un fils à un `CompteVirtuel`.
- virtual void `supprimerFils` (`CompteHierarchie` *_c)=0
Méthode qui sera redéfinie. Elle permettra de retirer un fils à un `CompteVirtuel`.
- virtual void `supprimerCompte` ()=0
Méthode virtuelle qui permet à l'utilisateur de supprimer le compte.
- virtual void `undo` (`Originator` *_o)=0
Méthode virtuelle qui permettra de restaurer un memento antérieur dans un compte réel.
- virtual void `saveState` (`Originator` *_o)=0
Méthode virtuelle qui permettra d'ajouter un memento à la liste de memento d'un compte réel.
- int `getId` () const noexcept
Méthode qui permet de retourner l'id d'un compte.
- virtual `Memento` * `getDernierMemento` ()=0
Méthode virtuelle qui permettra de retourner le dernier `Memento` enregistré d'un `CompteReel` (le solde actuel)
- virtual `Memento` * `getPremierMemento` ()=0
Méthode virtuelle qui permettra de retourner le premier `Memento` enregistré d'un `CompteReel` (le solde rapproché)
- virtual const list< `Memento` * > & `getMementos` () const =0
Méthode virtuelle qui permettra de récupérer la liste de tous les Mementos.

Fonctions membres protégées

- [CompteHierarchie](#) (string [_nom](#), int [_id](#)) noexcept
- virtual [~CompteHierarchie](#) ()=default

Destructeur.

Attributs protégés

- string [nom](#)
Le nom d'un Compte.
- set< [CompteHierarchie](#) * > [fils](#)
L'ensemble des fils d'un compte.
- const [PosteCompte](#) * [poste](#)
Pointeur sur un [PosteCompte](#) : [Racine](#), [Passif](#), [Recette](#), [Depense](#) ou [Actif](#).
- int [id](#)
L'id d'un compte. Il est unique et est géré automatiquement par [HierarchieBuilder](#).

Fonctions membres privées

- [CompteHierarchie](#) (const [CompteHierarchie](#) &[_c](#))=default
Constructeur par recopie privé.
- [CompteHierarchie](#) & [operator=](#) (const [CompteHierarchie](#) &[_c](#))=default
Surcharge de l'opérateur d'affectation privé.

Amis

- class [HierarchieBuilder](#)
- class [CompteManager](#)
- class [VisiteurFree](#)

4.2.1 Description détaillée

[CompteHierarchie](#) est la classe mère pour les éléments de la hiérarchie des comptes qui permet de manipuler tous les comptes via la même interface.

4.2.2 Documentation des constructeurs et destructeur

4.2.2.1 [CompteHierarchie](#)() [1/2]

```
CompteHierarchie::CompteHierarchie (
    string \_nom,
    int \_id ) [inline], [protected], [noexcept]
```

4.2.2.2 ~CompteHierarchie()

```
CompteHierarchie::~~CompteHierarchie ( ) [protected], [virtual], [default]
```

Destructeur.

4.2.2.3 CompteHierarchie() [2/2]

```
CompteHierarchie::CompteHierarchie (
    const CompteHierarchie & _c ) [private], [default]
```

Constructeur par recopie privé.

Paramètres

↔	: const CompteHierarchie &: référence constante sur le compte é recopier.
↔	
<code>c</code>	

4.2.3 Documentation des fonctions membres

4.2.3.1 `accepte()`

```
void CompteHierarchie::accepte (
    Visiteur & _v ) [pure virtual], [noexcept]
```

Méthode pour les visiteurs.

Paramètres

↔	: Visiteur &, visiteur à accepter.
↔	
<code>v</code>	

Implémenté dans [CompteReel](#), et [CompteVirtuel](#).

4.2.3.2 `ajouterFils()`

```
void CompteHierarchie::ajouterFils (
    CompteHierarchie * _c ) [pure virtual]
```

Methode qui sera redéfinie. Elle permettra d'ajouter un fils à un [CompteVirtuel](#).

Paramètres

↔	: CompteHierarchie *, pointeur sur le compte à ajouter
↔	
<code>c</code>	

Exceptions

<code>FLSR_EXC↔</code>	: impossible d'ajouter un fils à un compte réel.
<code>_H</code>	

Implémenté dans [CompteReel](#), et [CompteVirtuel](#).

4.2.3.3 getDernierMemento()

```
Memento * CompteHierarchie::getDernierMemento ( ) [pure virtual]
```

Méthode virtuelle qui permettra de retourner de retourner le dernier [Memento](#) enregistré d'un [CompteReel](#) (le solde actuel)

Exceptions

<code>MEMV_EXC↔ _C</code>	: Tentative d'utiliser un memento sur un compte virtuel
-------------------------------	---

Renvoie

Une pointeur sur le dernier memento de la liste

Implémenté dans [CompteVirtuel](#), et [CompteReel](#).

4.2.3.4 getFils()

```
const set< CompteHierarchie * > CompteHierarchie::getFils ( ) const [inline], [noexcept]
```

Methode qui permet d'obtenir les fils d'un compte.

Renvoie

Ensemble des fils d'un compte.

4.2.3.5 getId()

```
int CompteHierarchie::getId ( ) const [inline], [noexcept]
```

Méthode qui permet de retourner l'id d'un compte.

4.2.3.6 getMementos()

```
const list< Memento * > CompteHierarchie::getMementos ( ) const [pure virtual]
```

Méthode virtuelle qui permettra de récupérer la liste de tous les Mementos.

Exceptions

<code>MEMV_EXC↔ _C</code>	: Tentative d'utiliser un memento sur un compte virtuel
-------------------------------	---

Renvoie

Une liste non-modifiable de Mementos.

Implémenté dans [CompteVirtuel](#), et [CompteReel](#).

4.2.3.7 getNom()

```
string CompteHierarchie::getNom ( ) const [inline], [noexcept]
```

Méthode qui permet d'obtenir le nom du compte.

Renvoie

Le nom du compte.

4.2.3.8 getPoste()

```
Poste CompteHierarchie::getPoste ( ) const [inline], [noexcept]
```

Methode qui retourne le poste d'un compte.

Renvoie

RACINE ou DEPENSE ou RECETTE ou ACTIF ou PASSIF.

4.2.3.9 getPremierMemento()

```
Memento * CompteHierarchie::getPremierMemento ( ) [pure virtual]
```

Méthode virtuelle qui permettra de retourner de retourner le premier [Memento](#) enregistré d'un [CompteReel](#) (le solde rapproché)

Exceptions

<code>MEMV_EXC↔ _C</code>	: Tentative d'utiliser un memento sur un compte virtuel
-------------------------------	---

Renvoie

Une pointeur sur le premier memento de la liste

Implémenté dans [CompteVirtuel](#), et [CompteReel](#).

4.2.3.10 getType()

```
Type CompteHierarchie::getType ( ) const [pure virtual], [noexcept]
```

Methode qui retourne le type d'un Compte.

Renvoie

VIRTUEL si [CompteVirtual](#), REEL si [CompteReel](#).

Implémenté dans [CompteReel](#), et [CompteVirtual](#).

4.2.3.11 operator=()

```
CompteHierarchie & CompteHierarchie::operator= (
    const CompteHierarchie & _c ) [private], [default]
```

Surcharge de l'opérateur d'affectation privé.

Paramètres

↔	: const CompteHierarchie : référence constante sur le compte é affecter.
↔	
<i>c</i>	

4.2.3.12 saveState()

```
void CompteHierarchie::saveState (
    Originator _o ) [pure virtual]
```

Méthode virtuelle qui permettra d'ajouter un memento à la liste de memento d'un compte reel.

Paramètres

↔	: Originator*, pointeur sur un origiator qui va nous permettre de sauver le memento.
↔	
<i>o</i>	

Exceptions

<i>MEMV_EXC</i> ↔	: Tentative d'utiliser un memento sur un compte virtuel
<i>_C</i>	

Implémenté dans [CompteVirtual](#), et [CompteReel](#).

4.2.3.13 supprimerCompte()

```
void CompteHierarchie::supprimerCompte ( ) [pure virtual]
```

Méthode virtuelle qui permet à l'utilisateur de supprimer le compte.

Exceptions

<i>SUPREL_EXC_↔ _C</i>	le compte a des transactions il ne peut être supprimé
<i>SUPVIR_EXC_C</i>	les fils du compte n'ont pas été supprimés

Implémenté dans [CompteReel](#), et [CompteVirtuel](#).

4.2.3.14 supprimerFils()

```
void CompteHierarchie::supprimerFils (
    CompteHierarchie * _c ) [pure virtual]
```

Méthode qui sera redéfinie. Elle permettra de retirer un fils à un [CompteVirtuel](#).

Paramètres

<i>↔ _↔ C</i>	: CompteHierarchie *, pointeur sur le compte à retirer
-----------------------	--

Exceptions

<i>FLSR_EXC_↔ _H</i>	: impossible de retirer un fils à un compte réel.
--------------------------	---

Implémenté dans [CompteReel](#), et [CompteVirtuel](#).

4.2.3.15 toString()

```
string CompteHierarchie::toString ( ) const [pure virtual], [noexcept]
```

Méthode virtuelle qui retournera des informations sur les comptes.

Renvoie

Le nom du compte suivie de son type(virtuel) pour un [CompteVirtuel](#) et la première lettre du nom du poste du compte suivie du nom du compte pour un [CompteRéel](#).

Implémenté dans [CompteReel](#), et [CompteVirtuel](#).

4.2.3.16 undo()

```
void CompteHierarchie::undo (
    Originator * _o ) [pure virtual]
```

Méthode virtuelle qui permettra de restaurer un memento antérieur dans un compte réel.

Paramètres

\leftrightarrow	: Originator*, pointeur sur un originator qui va nous permettre de manipuler le memento.
$_ \leftrightarrow$	
o	

Exceptions

$MEMV_EXC \leftrightarrow$: Tentative d'utiliser un memento sur un compte virtuel
$_C$	

Implémenté dans [CompteVirtuel](#), et [CompteReel](#).

4.2.4 Documentation des fonctions amies et associées

4.2.4.1 CompteManager

```
friend class CompteManager [friend]
```

4.2.4.2 HierarchieBuilder

```
friend class HierarchieBuilder [friend]
```

4.2.4.3 VisiteurFree

```
friend class VisiteurFree [friend]
```

4.2.5 Documentation des données membres

4.2.5.1 fils

```
set< CompteHierarchie * > CompteHierarchie::fils [protected]
```

L'ensemble des fils d'un compte.

4.2.5.2 id

```
int CompteHierarchie::id [protected]
```

L'id d'un compte. Il est unique et est géré automatiquement par [HierarchieBuilder](#).

4.2.5.3 nom

```
string CompteHierarchie::nom [protected]
```

Le nom d'un Compte.

4.2.5.4 poste

```
PosteCompte * CompteHierarchie::poste [protected]
```

Pointeur sur un [PosteCompte](#) : [Racine](#), [Passif](#), [Recette](#), [Depense](#) ou [Actif](#).

La documentation de cette classe a été générée à partir du fichier suivant :

— [CompteHierarchie.h](#)

4.3 Référence de la classe `CompteManager`

[CompteManager](#) gère l'ensemble des Comptes.

```
#include <CompteManager.h>
```

Fonctions membres publiques

- `CompteHierarchie * getCompte (int _id)`
Méthode qui permet de renvoyer un pointeur sur un objet `CompteHierarchie` dont l'id a été passé en paramètre.
- `CompteHierarchie * getPere (CompteHierarchie * _c) const noexcept`
Retrouve le père d'un compte passé en argument.
- `void affiche () noexcept`
Méthode qui permet d'afficher la hiérarchie des comptes.
- `void ajouterFils (int _id_pere, const string & _nom_fils, Type _t, Poste _p)`
Méthode qui permet d'ajouter un fils à un `CompteVirtual`.
- `void supprimerCompte (int _id)`
Méthode qui permet de supprimer un Compte.
- `void deplacerFils (int _id_fils, int _id_futur_pere)`
Méthode qui permet de changer de place un compte dans la hiérarchie des comptes.
- `void charger ()`
Méthode qui permet de charger un fichier comptenant des comptes enregistrés lors des sessions précédente.
- `void setPath (const string & _file) noexcept`
Méthode qui permet de modifier l'attribut du path afin de renseigner un nouveau chemin d'un fichier où enregistrer les comptes ou à partir duquel sont restaurés les comptes enregistrés lors des sessions précédentes.
- `void sauver () const`
Méthode qui permet de sauver les comptes dans le fichier de chemin path.
- `void cloturer ()`
Méthode permettant de clôturer le livre : c'est à dire remettre à zéro les comptes de recette et de dépense par l'intermédiaire d'un compte `Résultat`.
- `void rapprocherCompte (int _id)`
Méthode qui permet le rapprochement d'un compte : c'est-à-dire fichier des transactions passées d'un compte.
- `double getSolde (int _id)`
Méthode qui permet d'obtenir le solde d'un compte.
- `void bilan (time_t _date)`
Méthode qui permet de faire le bilan, c'est à dire qui synthétise l'ensemble de ce que possède l'association (l'actif) et ce que l'association doit aux tiers (le passif) à la fin d'une période donnée, ainsi que le résultat sur cette période.
- `void releve (time_t _debut, time_t _fin)`
Méthode qui permet de faire le relevé des recettes et des dépenses, c'est à dire qui liste l'ensemble des dépenses et des recettes effectuées sur une période donnée.
- `void resultat ()`
Méthode qui permet de faire le résultat c'est à dire la différence entre le total des recettes et le total des dépenses.
- `void crediterCompte (int _id, double _montant)`
Méthode qui permet de créditer un compte Reel.
- `void debiterCompte (int _id, double _montant)`
Méthode qui permet de debiter un compte Reel.
- `Originator * getOriginator () noexcept`

Fonctions membres publiques statiques

- `static CompteManager & getInstance () noexcept`
Méthode qui permet d'obtenir l'instance `CompteManager`.

Fonctions membres privées

- `CompteManager () noexcept`
Constructeur privé
- `~CompteManager ()`
Destructeur privé
- `CompteManager (const CompteManager & _c)=default`
Constructeur par recopie, privé
- `CompteManager & operator= (const CompteManager & _c)=default`
Surcharge de l'opérateur d'affectation, privé

Attributs privés

- `CompteHierarchie * racine`
Pointeur sur le Compte `Racine` : le compte qui contient tous les autres comptes.
- `string path`
- `Originator * o = new Originator()`

Attributs privés statiques

— static `CompteManager` instance
instance `CompteManager`

4.3.1 Description détaillée

`CompteManager` gère l'ensemble des Comptes.

4.3.2 Documentation des constructeurs et destructeur

4.3.2.1 `CompteManager()` [1/2]

```
CompteManager::CompteManager ( ) [private], [noexcept]
```

Constructeur privé

4.3.2.2 `~CompteManager()`

```
CompteManager::~~CompteManager ( ) [private]
```

Destructeur privé

4.3.2.3 `CompteManager()` [2/2]

```
CompteManager::CompteManager (
    const CompteManager & _c ) [private], [default]
```

Constructeur par recopie, privé

Paramètres

<code>↔</code>	: const <code>CompteManager</code> &: référence constante sur le <code>CompteManager</code> à recopier.
<code>_↔</code>	
<code>c</code>	

4.3.3 Documentation des fonctions membres

4.3.3.1 affiche()

```
void CompteManager::affiche ( ) [noexcept]
```

Méthode qui permet d'afficher la hiérarchie des comptes.

4.3.3.2 ajouterFils()

```
void CompteManager::ajouterFils (
    int _id_pere,
    const string & _nom_fils,
    Type _t,
    Poste _p )
```

Méthode qui permet d'ajouter un fils à un [CompteVirtuel](#).

Paramètres

<i>_id_pere</i>	: int, id du CompteVirtuel auquel on veut ajouter un fils.
<i>_nom_fils</i>	: const string&, nom du compte fils que l'on veut ajouter.
<i>_t</i>	: const string&, type du compte fils que l'on veut ajouter : VIRTUEL ou REEL.
<i>_p</i>	: Poste, poste du compte fils que l'on veut ajouter (DEPENSE, RECETTE, ACTIF, PASSIF).

Exceptions

<i>TYPE_EXC_H</i>	Type de compte inconnu
<i>POSTE_EXC_H</i>	Type de poste inconnu
<i>RULE_EXC_H</i>	Impossible d'ajouter ce compte fils à ce compte

4.3.3.3 bilan()

```
void CompteManager::bilan (
    time_t _date )
```

Méthode qui permet de faire le bilan, c'est à dire qui synthétise l'ensemble de ce que possède l'association (l'actif) et ce que l'association doit aux tiers (le passif) à la fin d'une période donnée, ainsi que le résultat sur cette période.

Paramètres

<i>_date</i>	: time_t , la date à laquelle faire le bilan
--------------	--

4.3.3.4 charger()

```
void ComptManager::charger ( )
```

Méthode qui permet de charger un fichier contenant des comptes enregistrés lors des sessions précédente.

Exceptions

<i>non</i>	implémentée
------------	-------------

4.3.3.5 cloturer()

```
void ComptManager::cloturer ( )
```

Méthode permettant de clôturer le livre : c'est à dire remettre à zéro les comptes de recette et de dépense par l'intermédiaire d'un compte Résultat.

4.3.3.6 crediterCompte()

```
void ComptManager::crediterCompte (
    int _id,
    double _montant )
```

Méthode qui permet de créditer un compte Reel.

Paramètres

<i>_id</i>	: int, id du compte à créditer.
<i>_montant</i>	: double, montant à créditer sur le compte.

Exceptions

<i>IDNF_EXC_H</i>	: l'ID recherché n'existe pas.
<i>MEMV_EXC↔_C</i>	: Tentative d'utiliser un memento sur un compte virtuel
<i>ROOT_EXC↔_C</i>	Lancement d'ExceptionComptabilité : Impossible de crediter le compte racine.

4.3.3.7 debiterCompte()

```
void ComptManager::debiterCompte (
    int _id,
    double _montant )
```

Methode qui permet de debiter un compte Reel.

Paramètres

<i>_id</i>	: int, id du compte à debiter.
<i>_montant</i>	: double, montant à débiter sur le compte.

Exceptions

<i>IDNF_EXC_H</i>	: l'ID recherché n'existe pas.
<i>MEMV_EXC↔ _C</i>	: Tentative d'utiliser un memento sur un compte virtuel
<i>ROOT_EXC↔ _C</i>	Lancement d'ExceptionComptabilité : Impossible de débiter le compte racine.

4.3.3.8 deplacerFils()

```
void CompteManager::deplacerFils (
    int _id_fils,
    int _id_futur_pere )
```

Methode qui permet de changer de place un compte dans la hiérarchie des comptes.

Paramètres

<i>_id_futur_pere</i>	: int, id du compte sous lequel on veut déplacer le compte (c'est à dire le nouveau père du compte).
<i>_id_fils</i>	: int, id du compte à déplacer.

Exceptions

<i>IDNF_EXC_H</i>	l'ID recherché n'existe pas.
<i>RULE_EXC↔ _H</i>	si le futur père n'est pas du bon type.
<i>FLSR_EXC↔ _H</i>	si le futur père est réel.
<i>FLSR_EXC↔ _H</i>	impossible de retirer un fils à un compte réel
<i>FLSR_EXC↔ _H</i>	impossible d'ajouter un fils à un compte réel

4.3.3.9 getCompte()

```
CompteHierarchie * CompteManager::getCompte (
    int _id )
```


Méthode qui permet de renvoyer un pointeur sur un objet `CompteHierarchie` dont l'id a été passé en paramètre.

Paramètres

<code>↔</code> <code>_↔</code> <code>id</code>	: int id du compte sur lequel on veut obtenir un pointeur.
--	--

Renvoie

Un pointeur sur le compte recherché s'il existe.

Exceptions

<code>IDNF_EXC↔</code> <code>_H</code>	: l'ID recherché n'existe pas.
---	--------------------------------

4.3.3.10 `getInstance()`

```
static CompteManager & CompteManager::getInstance ( ) [inline], [static], [noexcept]
```

Méthode qui permet d'obtenir l'instance `CompteManager`.

Renvoie

instance

4.3.3.11 `getOriginator()`

```
Originator* CompteManager::getOriginator ( ) [inline], [noexcept]
```

4.3.3.12 `getPere()`

```
CompteHierarchie * CompteManager::getPere (   
    CompteHierarchie * _c ) const [noexcept]
```

Retrouve le père d'un compte passé en argument.

Paramètres

<code>↔</code> <code>_↔</code> <code>c</code>	: const <code>CompteHierarchie*</code> pointeur sur le fils à trouver
---	---

Renvoie

Un pointeur sur le compte père s'il existe, sinon nullptr

4.3.3.13 getSolde()

```
double CompteManager::getSolde (
    int _id )
```

Méthode qui permet d'obtenir le solde d'un compte.

Paramètres

↔	: int id du compte pour lequel on veut obtenir le solde.
↔	
<i>id</i>	

Exceptions

<i>IDNF_EXC↔</i>	: l'ID recherché n'existe pas.
<i>_H</i>	

Renvoie

Le solde du compte dont l'id a été passé en paramètre si le compte existe.

4.3.3.14 operator=()

```
CompteManager & CompteManager::operator= (
    const CompteManager & _c ) [private], [default]
```

Surcharge de l'opérateur d'affectation, privé

Paramètres

↔	: const CompteManager & : référence constante sur le CompteManager à affecter.
↔	
<i>c</i>	

4.3.3.15 rapprocherCompte()

```
void CompteManager::rapprocherCompte (
    int _id )
```

Méthode qui permet le rapprochement d'un compte : c'est-à-dire fichier des transactions passées d'un compte.

Paramètres

<code>↔</code> <code>↔</code> <code>↔</code> <code>id</code>	: int, id du compte sur lequel on veut faire le rapprochement.
---	--

Exceptions

<code>MEMV_EXC↔</code> <code>_C</code>	: Tentative d'utiliser un memento sur un compte virtuel
<code>IDNF_EXC_H</code>	: l'ID recherché n'existe pas.

4.3.3.16 `releve()`

```
void CompteManager::releve (
    time_t _debut,
    time_t _fin )
```

Méthode qui permet de faire le relevé des recettes et des dépenses, c'est à dire qui liste l'ensemble des dépenses et des recettes effectuées sur une période donnée.

Paramètres

<code>_debut</code>	: <code>time_t</code> , le début de la période sur laquelle faire le relevé
<code>_fin</code>	: <code>time_t</code> , la fin de la période sur laquelle faire le relevé

4.3.3.17 `resultat()`

```
void CompteManager::resultat ( )
```

Méthode qui permet de faire le résultat c'est à dire la différence entre le total des recettes et le total des dépenses.

4.3.3.18 `sauver()`

```
void CompteManager::sauver ( ) const
```

Méthode qui permet de sauver les comptes dans le fichier de chemin path.

Exceptions

<code>non</code>	implémentée
------------------	-------------

4.3.3.19 setPath()

```
void CompteManager::setPath (
    const string & _file ) [noexcept]
```

Méthode qui permet de modifier l'attribut du path afin de renseigner un nouveau chemin d'un fichier où enregistrer les comptes ou à partir duquel sont restaurés les comptes enregistrés lors des sessions précédentes.

Paramètres

<code>_file</code>	: cosnt string&, nouveau chemin du fichier.
--------------------	---

4.3.3.20 supprimerCompte()

```
void CompteManager::supprimerCompte (
    int _id )
```

Méthode qui permet de supprimer un Compte.

Paramètres

<code>↔ _↔ id</code>	: int, id du compte que l'on veut supprimer.
------------------------------	--

Exceptions

<code>IDNF_EXC_H</code>	l'ID recherché n'existe pas.
<code>SUPREL_EXC_↔ _C</code>	le compte a des transactions il ne peut être supprimé
<code>SUPVIR_EXC_C</code>	les fils du compte n'ont pas été supprimés
<code>RULE_EXC_H</code>	le compte est nécessaire au fonctionnement de l'application et ne peut être supprimé

4.3.4 Documentation des données membres

4.3.4.1 instance

```
CompteManager CompteManager::instance [static], [private]
```

instance [CompteManager](#)

4.3.4.2 o

```
Originator* CompteManager::o = new Originator() [private]
```

4.3.4.3 path

```
string CompteManager::path [private]
```

4.3.4.4 racine

```
CompteHierarchie * CompteManager::racine [private]
```

Pointeur sur le Compte [Racine](#) : le compte qui contient tous les autres comptes.

La documentation de cette classe a été générée à partir du fichier suivant :

- [CompteManager.h](#)
- [CompteManager.cpp](#)

4.4 Référence de la classe CompteReel

[CompteReel](#) représente les compte réels (les comptes sur lesquels auront lieu les opérations).

```
#include <CompteReel.h>
```

Est dérivée de [CompteHierarchie](#).

Fonctions membres publiques

- string [toString](#) () const noexcept override
Redéfinition de la méthode [toString\(\)](#) de [CompteHiérarchie](#).
- Type [getType](#) () const noexcept override
Redéfinition de la méthode [getType](#) de [CompteHierarchie](#) afin qu'elle retourne le type d'un compte réel.
- void [accepte](#) (Visiteur &_v) noexcept override
Redéfinition de la méthode [accepte\(Visiteur& _v\)](#) de [CompteHierarchie](#).
- void [ajouterFils](#) (CompteHierarchie *_c) override
Redéfinition de [ajouterFils\(CompteHierarchie _c\)](#) de [CompteHiérarchie](#). Cette méthode ne fait rien sur [CompteReel](#) car il n'a pas de fils.*
- void [supprimerFils](#) (CompteHierarchie *_c) override
Méthode qui sera redéfinie. Elle permettra de retirer un fils à un [CompteVirtuel](#).
- void [supprimerCompte](#) () override
Redéfinition de [supprimerCompte](#) de [CompteHierarchie](#). Permet de supprimer un compte si ses transactions le permettent.
- void [undo](#) (Originator *_o) noexcept override
Méthode qui permet de restaurer un memento antérieur.
- void [saveState](#) (Originator _o) noexcept override
Méthode qui permet d'ajouter un memento à la liste de memento.
- Memento * [getDernierMemento](#) () noexcept override
Méthode virtuelle qui permettra de retourner de retourner le dernier [Memento](#) enregistré d'un [CompteReel](#) (le solde actuel)
- Memento * [getPremierMemento](#) () noexcept override
Méthode qui permettra de retourner de retourner le premier [Memento](#) enregistré (le solde rapproché)
- const list< Memento * > & [getMementos](#) () const noexcept override
Méthode qui permet de récupérer la liste de tous les Mementos.

Fonctions membres privées

- [CompteReel](#) (const string &_nom, int _id) noexcept
- [~CompteReel](#) () noexcept
Destructeur privé
- [CompteReel](#) (const [CompteReel](#) &_c)=default
Constructeur par recopie privé
- [CompteReel](#) & [operator=](#) (const [CompteReel](#) &_c)=default
Surcharge de l'opérateur d'affectation privé.

Attributs privés

- list< [Memento](#) * > [mementos](#)
Liste de mementos qui permet de garder en mémoire les soldes des derniers rapprochements.

Amis

- class [HierarchieBuilder](#)
- class [CompteManager](#)
- class [VisiteurFree](#)

Membres hérités additionnels

4.4.1 Description détaillée

[CompteReel](#) représente les compte réels (les comptes sur lesquels auront lieu les opérations).

4.4.2 Documentation des constructeurs et destructeur

4.4.2.1 [CompteReel\(\)](#) [1/2]

```
CompteReel::CompteReel (
    const string & _nom,
    int _id ) [inline], [private], [noexcept]
```

4.4.2.2 [~CompteReel\(\)](#)

```
CompteReel::~~CompteReel ( ) [private], [noexcept]
```

Destructeur privé

4.4.2.3 [CompteReel\(\)](#) [2/2]

```
CompteReel::CompteReel (
    const CompteReel & _c ) [private], [default]
```

Constructeur par recopie privé

Paramètres

<code>↔</code>	: const CompteReel &: référence constante sur le compte réel à recopier.
<code>_↔</code>	
<code>c</code>	

4.4.3 Documentation des fonctions membres

4.4.3.1 `accepte()`

```
void CompteReel::accepte (
    Visiteur & _v ) [override], [virtual], [noexcept]
```

Redéfinition de la méthode [accepte\(Visiteur& _v\)](#) de [CompteHierarchie](#).

Paramètres

<code>↔</code>	: Visiteur &, visiteur à accepter
<code>_↔</code>	
<code>v</code>	

Implémente [CompteHierarchie](#).

4.4.3.2 `ajouterFils()`

```
void CompteReel::ajouterFils (
    CompteHierarchie * _c ) [override], [virtual]
```

Redéfinition de [ajouterFils\(CompteHierarchie* _c\)](#) de [CompteHiérarchie](#). Cette méthode ne fait rien sur [CompteReel](#) car il n'a pas de fils.

Exceptions

<code>FLSR_EXC↔</code> <code>_H</code>	: impossible d'ajouter un fils à un compte réel.
---	--

Implémente [CompteHierarchie](#).

4.4.3.3 `getDernierMemento()`

```
Memento* CompteReel::getDernierMemento ( ) [inline], [override], [virtual], [noexcept]
```

Méthode virtuelle qui permettra de retourner de retourner le dernier [Memento](#) enregistré d'un [CompteReel](#) (le solde actuel)

Exceptions

<code>MEMV_EXC↔ _C</code>	: Tentative d'utiliser un memento sur un compte virtuel
-------------------------------	---

Renvoie

Une pointeur sur le dernier memento de la liste

Implémente [CompteHierarchie](#).

4.4.3.4 getMementos()

```
const list< Memento * > & CompteReel::getMementos ( ) const [inline], [override], [virtual], [noexcept]
```

Méthode qui permet de récupérer la liste de tous les Mementos.

Renvoie

Une liste non-modifiable de Mementos.

Implémente [CompteHierarchie](#).

4.4.3.5 getPremierMemento()

```
Memento * CompteReel::getPremierMemento ( ) [inline], [override], [virtual], [noexcept]
```

Méthode qui permettra de retourner de retourner le premier [Memento](#) enregistré (le solde rapproché)

Renvoie

Une pointeur sur le premier memento de la liste

Implémente [CompteHierarchie](#).

4.4.3.6 getType()

```
Type CompteReel::getType ( ) const [inline], [override], [virtual], [noexcept]
```

Redéfinition de la méthode `getType` de [CompteHierarchie](#) afin qu'elle retourne le type d'un compte réel.

Renvoie

REEL

Implémente [CompteHierarchie](#).

4.4.3.7 operator=()

```
CompteReel & CompteReel::operator= (
    const CompteReel & _c ) [private], [default]
```

Surcharge de l'opérateur d'affectation privé.

Paramètres

↔	: const CompteReel & : référence constante sur le compte réel à affecter.
↔	
<i>c</i>	

4.4.3.8 saveState()

```
void CompteReel::saveState (
    Originator _o ) [inline], [override], [virtual], [noexcept]
```

Méthode qui permet d'ajouter un memento à la liste de memento.

Paramètres

↔	: Originator*, pointeur sur un originator qui va nous permettre de sauver le memento.
↔	
<i>o</i>	

Implémente [CompteHierarchie](#).

4.4.3.9 supprimerCompte()

```
void CompteReel::supprimerCompte ( ) [override], [virtual]
```

Redéfinition de supprimerCompte de [CompteHierarchie](#). Permet de supprimer un compte si ses transactions le permettent.

Exceptions

<i>SUPREL_EXC↔</i>	le compte a des transactions il ne peut être supprimé
<i>_C</i>	

Implémente [CompteHierarchie](#).

4.4.3.10 supprimerFils()

```
void CompteReel::supprimerFils (
    CompteHierarchie * _c ) [override], [virtual]
```

Methode qui sera redéfinie. Elle permettra de retirer un fils à un [CompteVirtuel](#).

Paramètres

↔	: CompteHierarchie*, pointeur sur le compte à retirer
↔	
c	

Exceptions

FLSR_EXC↔ _H	: impossible de retirer un fils à un compte réel.
-----------------	---

Implémente [CompteHierarchie](#).

4.4.3.11 toString()

```
string CompteReel::toString ( ) const [override], [virtual], [noexcept]
```

Redéfinition de la méthode [toString\(\)](#) de CompteHiérarchie.

Renvoie

Première lettre de du nom du poste du compte (D pour dépense, R pour recette, A pour actif ou P pour passif) suivie de REEL.

Implémente [CompteHierarchie](#).

4.4.3.12 undo()

```
void CompteReel::undo (
    Originator * _o ) [override], [virtual], [noexcept]
```

Méthode qui permet de restaurer un memento antérieur.

Paramètres

↔	: Originator*, pointeur sur un originator qui va nous permettre de manipuler le memento.
↔	
o	

Implémente [CompteHierarchie](#).

4.4.4 Documentation des fonctions amies et associées

4.4.4.1 CompteManager

```
friend class CompteManager [friend]
```

4.4.4.2 HierarchieBuilder

```
friend class HierarchieBuilder [friend]
```

4.4.4.3 VisiteurFree

```
friend class VisiteurFree [friend]
```

4.4.5 Documentation des données membres

4.4.5.1 mementos

```
liste< Memento * > CompteReel::mementos [private]
```

Liste de mementos qui permet de garder en mémoire les soldes des derniers rapprochements.

La documentation de cette classe a été générée à partir du fichier suivant :

- [CompteReel.h](#)
- [CompteReel.cpp](#)

4.5 Référence de la classe [CompteVirtuel](#)

[CompteVirutel](#) représente les [Compte Virtuels](#) (comptes destinés à regrouper des sous-compte sans enregistrer d'opération).

```
#include <CompteVirtuel.h>
```

Est dérivée de [CompteHierarchie](#).

Fonctions membres publiques

- string [toString](#) () const noexcept override
Redéfinition de la méthode [toString\(\)](#) de [CompteHiérarchie](#).
- Type [getType](#) () const noexcept override
Redéfinition de la méthode [getType](#) de [CompteHiérarchie](#) afin qu'elle retourne le type d'un compte virtuel.
- void [accepte](#) ([Visiteur](#) &_v) noexcept override
Redéfinition de la méthode [accepte\(Visiteur& _v\)](#) de [CompteHiérarchie](#).
- void [ajouterFils](#) ([CompteHiérarchie](#) *_c) noexcept override
Redéfinition de [ajouterFils\(CompteHiérarchie _c\)](#) de [CompteHiérarchie](#). Cette méthode permet d'ajouter un fils d'un [CompteVirtuel](#).*
- void [supprimerFils](#) ([CompteHiérarchie](#) *_c) noexcept override
Méthode qui sera redéfinie. Elle permettra de retirer un fils à un [CompteVirtuel](#).
- void [supprimerCompte](#) () override
Redéfinition de [supprimerCompte\(\)](#) de [CompteHiérarchie](#).
- void [undo](#) ([Originator](#) *_o) override
Méthode virtuelle undo redéfinie de [CompteHiérarchie](#). On ne peut pas utiliser cette méthode avec un compte virtuel : lancement d'une exception.
- void [saveState](#) ([Originator](#) _o) override
Méthode virtuelle qui permettra d'ajouter un memento à la liste de memento d'un compte reel.
- [Memento](#) * [getDernierMemento](#) () override
Genère une exception car ne peut être lancé que pour un compte reel.
- [Memento](#) * [getPremierMemento](#) () override
Genère une exception car ne peut être lancé que pour un compte reel.
- const list< [Memento](#) * > & [getMementos](#) () const override
Genère une exception car ne peut être lancé que pour un compte reel.

Fonctions membres privées

- [CompteVirtuel](#) (const string &_nom, int _id) noexcept
- [~CompteVirtuel](#) ()=default
Destructeur privé
- [CompteVirtuel](#) (const [CompteVirtuel](#) &_c)=default
Constructeur par recopie privé
- [CompteVirtuel](#) & [operator=](#) (const [CompteVirtuel](#) &_c)=default
Surcharge de l'opérateur d'affectation privé

Amis

- class [HierarchieBuilder](#)
- class [CompteManager](#)
- class [VisiteurFree](#)

Membres hérités additionnels

4.5.1 Description détaillée

CompteVirutel représente les Compte Virtuels (comptes destinés à regrouper des sous-compte sans enregistrer d'opération).

4.5.2 Documentation des constructeurs et destructeur

4.5.2.1 `CompteVirtuel()` [1/2]

```
CompteVirtuel::CompteVirtuel (
    const string & _nom,
    int _id ) [inline], [private], [noexcept]
```

4.5.2.2 `~CompteVirtuel()`

```
CompteVirtuel::~~CompteVirtuel ( ) [private], [default]
```



Destructeur privé

4.5.2.3 `CompteVirtuel()` [2/2]

```
CompteVirtuel::CompteVirtuel (
    const CompteVirtuel & _c ) [private], [default]
```

Constructeur par recopie privé

Paramètres

	: const CompteVirtuel &: référence constante sur le compte virtuel à recopier.
	
c	



4.5.3 Documentation des fonctions membres

4.5.3.1 `accepte()`

```
void CompteVirtuel::accepte (
    Visiteur & _v ) [override], [virtual], [noexcept]
```

Redéfinition de la méthode `accepte(Visiteur& _v)` de [CompteHierarchie](#).

Paramètres

	: Visiteur &, visiteur à accepter.
	
v	

Implémente [CompteHierarchie](#).

4.5.3.2 ajouterFils()

```
void CompteVirtuel::ajouterFils (
    CompteHierarchie * _c ) [inline], [override], [virtual], [noexcept]
```

Redéfinition de [ajouterFils\(\[CompteHierarchie\]\(#\)* _c\)](#) de [CompteHiérarchie](#). Cette méthode permet d'ajouter un fils d'un [CompteVirtuel](#).

Paramètres

\leftrightarrow	: CompteHierarchie *, pointeur sur le compte à ajouter.
$_ \leftrightarrow$	
$_C$	

Implémente [CompteHierarchie](#).

4.5.3.3 getDernierMemento()

```
Memento * CompteVirtuel::getDernierMemento ( ) [override], [virtual]
```

Genère une exception car ne peut être lancé que pour un compte reel.

Exceptions

$MEMV_EXC_{\leftrightarrow}$ $_C$: Tentative d'utiliser un memento sur un compte virtuel
--	---

Implémente [CompteHierarchie](#).

4.5.3.4 getMementos()

```
const list< Memento * > & CompteVirtuel::getMementos ( ) const [override], [virtual]
```

Genère une exception car ne peut être lancé que pour un compte reel.

Exceptions

$MEMV_EXC_{\leftrightarrow}$ $_C$: Tentative d'utiliser un memento sur un compte virtuel
--	---

Implémente [CompteHierarchie](#).

4.5.3.5 getPremierMemento()

```
Memento * CompteVirtuel::getPremierMemento ( ) [override], [virtual]
```

Genère une exception car ne peut être lancé que pour un compte reel.

Exceptions

<code>MEMV_EXC↔ _C</code>	: Tentative d'utiliser un memento sur un compte virtuel
-------------------------------	---

Implémente [CompteHierarchie](#).

4.5.3.6 getType()

```
Type CompteVirtuel::getType ( ) const [inline], [override], [virtual], [noexcept]
```

Redéfinition de la méthode getType de [CompteHierarchie](#) afin qu'elle retourne le type d'un compte virtuel.

Renvoie

VIRTUEL

Implémente [CompteHierarchie](#).

4.5.3.7 operator=()

```
CompteVirtuel & CompteVirtuel::operator= (
    const CompteVirtuel & _c ) [private], [default]
```

Surcharge de l'opérateur d'affectation privé

Paramètres

<code>↔ _↔ c</code>	: const CompteVirtuel & : référence constante sur le compte virtuel à affecter.
-----------------------------	---

4.5.3.8 saveState()

```
void CompteVirtuel::saveState (
    Originator _o ) [override], [virtual]
```

Méthode virtuelle qui permettra d'ajouter un memento à la liste de memento d'un compte reel.

Paramètres

\leftrightarrow	: Originator*, pointeur sur un origiator qui va nous permettre de sauver le memento.
$_ \leftrightarrow$	
<i>o</i>	

Exceptions

<i>MEMV_EXC</i> $_ \leftrightarrow$ <i>_C</i>	: Tentative d'utiliser un memento sur un compte virtuel
--	---

Implémente [CompteHierarchie](#).

4.5.3.9 supprimerCompte()

```
void CompteVirtuel::supprimerCompte ( ) [override], [virtual]
```

Redéfinition de [supprimerCompte\(\)](#) de [CompteHiérarchie](#).

Exceptions

<i>SUPVIR_EXC</i> $_ \leftrightarrow$ <i>_C</i>	les fils du compte n'ont pas été supprimés
--	--

Implémente [CompteHierarchie](#).

4.5.3.10 supprimerFils()

```
void CompteVirtuel::supprimerFils (
    CompteHierarchie * \_c ) [inline], [override], [virtual], [noexcept]
```

Methode qui sera redéfinie. Elle permettra de retirer un fils à un [CompteVirtuel](#).

Paramètres

\leftrightarrow	: CompteHierarchie *, pointeur sur le compte à retirer
$_ \leftrightarrow$	
<i>c</i>	

Exceptions

<i>FLSR_EXC</i> $_ \leftrightarrow$ <i>_H</i>	: impossible de retirer un fils à un compte réel.
--	---

Implémente [CompteHierarchie](#).

4.5.3.11 toString()

```
std::string CompteVirtuel::toString ( ) const [override], [virtual], [noexcept]
```

Redéfinition de la méthode [toString\(\)](#) de [CompteHiérarchie](#).

Renvoie

Le nom du compte, suivie de virtuel.

Implémente [CompteHierarchie](#).

4.5.3.12 undo()

```
void CompteVirtuel::undo (
    Originator * _o ) [override], [virtual]
```

Méthode virtuelle undo redéfinie de [CompteHierarchie](#). On ne peut pas utiliser cette méthode avec un compte virtuel : lancement d'une exception.

Exceptions

<code>MEMV_EXC↔ _C</code>	: Tentative d'utiliser un memento sur un compte virtuel
-------------------------------	---

Implémente [CompteHierarchie](#).

4.5.4 Documentation des fonctions amies et associées

4.5.4.1 CompteManager

```
friend class CompteManager [friend]
```

4.5.4.2 HierarchieBuilder

```
friend class HierarchieBuilder [friend]
```

4.5.4.3 VisiteurFree

```
friend class VisiteurFree [friend]
```

La documentation de cette classe a été générée à partir du fichier suivant :

- [CompteVirtuel.h](#)
- [CompteVirtuel.cpp](#)

4.6 Référence de la classe **Depense**

[Depense](#) permet de représenter les propriétés des comptes de poste [Depense](#).

```
#include <Depense.h>
```

Est dérivée de [PosteAD](#).

Fonctions membres publiques

- [Poste](#) [getPoste](#) () const noexcept override
Redéfinition de la méthode virtuelle [getPoste](#) de [PosteCompte](#). Cette méthode retourne le nom du poste [Depense](#): [DEPENDSE](#).
- string [getLetter](#) () const noexcept override
Redéfinition de la méthode virtuelle [getLetter](#) de [PosteCompte](#). Cette méthode renvoie la première lettre d'un poste [Depense](#) : [D](#).

Fonctions membres publiques statiques

- static const [PosteCompte](#) & [getInstance](#) () noexcept
Méthode qui permet d'obtenir l'instance [Depense](#).

Fonctions membres privées

- [Depense](#) ()=default
Constructeur privé
- [~Depense](#) ()=default
Destructeur privé
- [Depense](#) (const [Depense](#) &_d)=default
Constructeur par copie privé.
- [Depense](#) & [operator=](#) (const [Depense](#) &_d)=default
Surcharge de l'opérateur d'affectation privé.

Attributs privés statiques

- static [Depense](#) [instance](#)
instance [Depense](#)

4.6.1 Description détaillée

[Depense](#) permet de représenter les propriétés des comptes de poste [Depense](#).

4.6.2 Documentation des constructeurs et destructeur

4.6.2.1 Depense() [1/2]

```
Depense::Depense ( ) [private], [default]
```

Constructeur privé

4.6.2.2 ~Depense()

```
Depense::~Depense ( ) [private], [default]
```

Destructeur privé

4.6.2.3 Depense() [2/2]

```
Depense::Depense (
    const Depense & _d ) [private], [default]
```

Constructeur par recopie privé.

Paramètres

↩	: const Depense &, référence constante sur le poste dépense à recopier.
↩ _↩ d	

4.6.3 Documentation des fonctions membres

4.6.3.1 getInstance()

```
static const PosteCompte & Depense::getInstance ( ) [inline], [static], [noexcept]
```

Méthode qui permet d'obtenir l'instance [Depense](#).

Renvoie

instance

4.6.3.2 getLetter()

```
string Depense::getLetter ( ) const [inline], [override], [virtual], [noexcept]
```

Redéfinition de la méthode virtuelle getLetter de [PosteCompte](#). Cette méthode renvoie la première lettre d'un poste [Depense](#) : D.

Renvoie

D

Implémente [PosteCompte](#).

4.6.3.3 getPoste()

```
Poste Depense::getPoste ( ) const [inline], [override], [virtual], [noexcept]
```

Redéfinition de la méthode virtuelle getPoste de [PosteCompte](#). Cette méthode retourne le nom du poste [Depense](#): DEPENSE.

Renvoie

DEPENSE

Implémente [PosteCompte](#).

4.6.3.4 operator=()

```
Depense & Depense::operator= (
    const Depense & _d ) [private], [default]
```

Surcharge de l'opérateur d'affectation privé.

Paramètres

←	: const Depense &, référence constante sur le poste dépense à affecter.
←	
<i>d</i>	

4.6.4 Documentation des données membres

4.6.4.1 instance

```
Depense Depense::instance [static], [private]
```

instance [Depense](#)

La documentation de cette classe a été générée à partir du fichier suivant :

- [Depense.h](#)
- [Depense.cpp](#)

4.7 Référence de la classe Exception

[Exception](#) permet de gérer une erreur ainsi que son message.

```
#include <Exception.h>
```

Dérivée par [ExceptionComptabilite](#), [ExceptionFichier](#), [ExceptionHierarchie](#), et [ExceptionTransaction](#).

Fonctions membres publiques

- const string & [getMessage](#) () const noexcept
Méthode qui permet d'obtenir le message d'erreur.
- int const [getCode](#) () const noexcept
Méthode qui permet d'obtenir le code d'erreur en vue de son traitement.

Fonctions membres protégées

- [Exception](#) (string _s, int _code_erreur) noexcept
Constructeur.

Attributs privés

- string [message](#)
Explication des raisons de la création de l'exception.
- int [code_erreur](#)
Code permettant l'identification de l'erreur en vue de son traitement.

4.7.1 Description détaillée

[Exception](#) permet de gérer une erreur ainsi que son message.

4.7.2 Documentation des constructeurs et destructeur

4.7.2.1 Exception()

```
Exception::Exception (  
    string _s,  
    int _code_erreur ) [inline], [protected], [noexcept]
```

Constructeur.

4.7.3 Documentation des fonctions membres

4.7.3.1 getCode()

```
const string & Exception::getCode ( ) const [inline], [noexcept]
```

Méthode qui permet d'obtenir le code d'erreur en vue de son traitement.

Renvoie

Le code d'erreur.

4.7.3.2 getMessage()

```
const string & Exception::getMessage ( ) const [inline], [noexcept]
```

Méthode qui permet d'obtenir le message d'erreur.

Renvoie

Le message d'erreur.

4.7.4 Documentation des données membres

4.7.4.1 code_erreur

```
int Exception::code_erreur [private]
```

Code permettant l'identification de l'erreur en vue de son traitement.

4.7.4.2 message

```
string Exception::message [private]
```

Explication des raisons de la création de l'exception.

La documentation de cette classe a été générée à partir du fichier suivant :

— [Exception.h](#)

4.8 Référence de la classe ExceptionComptabilite

[ExceptionComptabilite](#) est créée lors d'une opération de comptabilité illégale.

```
#include <ExceptionCompte.h>
```

Est dérivée de [Exception](#).

Types publics

```
— enum CodeExcepComptabilite {  
    UKWN_EXC_C = 20, MEMV_EXC_C = 21, ROOT_EXC_C = 22, SUPVIR_EXC_C =23,  
    SUPREL_EXC_C =24 }
```

Fonctions membres publiques

```
— ExceptionComptabilite (string _s, CodeExcepComptabilite _code_erreur=CodeExcepComptabilite::UKW↵  
    N_EXC_C) noexcept  
    Constructeur à partir d'un message d'erreur.
```

Membres hérités additionnels

4.8.1 Description détaillée

[ExceptionComptabilite](#) est créée lors d'une opération de comptabilité illégale.

4.8.2 Documentation des énumérations membres

4.8.2.1 CodeExcepComptabilite

```
enum ExceptionComptabilite::CodeExcepComptabilite
```

Valeurs énumérées

UKWN_EXC_C	20 : une exception inconnue
MEMV_EXC_C	21 : Tentative de manipuler les mementos d'un CompteVirtuel .
ROOT_EXC_C	22 : Tentative de manipulation du solde sur la racine
SUPVIR_EXC_C	23 : Tentative de suppression d'un compte virtuel qui a des fils
SUPREL_EXC_C	24 : Tentative de suppression d'un compte reel qui a des transactions

4.8.3 Documentation des constructeurs et destructeur

4.8.3.1 ExceptionComptabilite()

```
ExceptionComptabilite::ExceptionComptabilite (
    string _s,
    CodeExcepComptabilite _code_erreur = CodeExcepComptabilite::UKWN_EXC_C ) [inline],
[noexcept]
```

Constructeur à partir d'un message d'erreur.

La documentation de cette classe a été générée à partir du fichier suivant :

— [ExceptionCompte.h](#)

4.9 Référence de la classe ExceptionFichier

traite les exceptions de la partie fichier

```
#include <exceptiontransaction.h>
```

Est dérivée de [Exception](#).

Types publics

— enum [CodeExcepFichier](#) { [UKWN_EXC_F](#) = 60, [UFND_EXC_F](#) = 61, [SNTX_EXC_F](#) = 62 }

Fonctions membres publiques

— [ExceptionFichier](#) (string _message, [CodeExcepFichier](#) _code_erreur=[CodeExcepFichier::UKWN_EXC_F](#)) noexcept
Constructeur, hérite de [Exception](#).

Membres hérités additionnels

4.9.1 Description détaillée

traite les exceptions de la partie fichier

4.9.2 Documentation des énumérations membres

4.9.2.1 CodeExcepFichier

```
enum ExceptionFichier::CodeExcepFichier
```

Valeurs énumérées

UKWN_EXC↔ _F	60 : une exception inconnue
UFND_EXC_F	61 : le fichier est introuvable ou inaccessible
SNTX_EXC_F	62 : le fichier présente une syntaxe incorrecte

4.9.3 Documentation des constructeurs et destructeur

4.9.3.1 ExceptionFichier()

```
ExceptionFichier::ExceptionFichier (
    string _message,
    CodeExcepFichier _code_erreur = CodeExcepFichier::UKWN_EXC_F ) [inline], [explicit],
[noexcept]
```

Constructeur, hérite de [Exception](#).

Paramètres

<code>_message</code>	: string, message d'erreur expliquant l'exception
<code>_code_erreur</code>	: CodeExcepFichier, code d'erreur permettant le traitement de l'exception

La documentation de cette classe a été générée à partir du fichier suivant :

— [exceptiontransaction.h](#)

4.10 Référence de la classe ExceptionHierarchie

[ExceptionHierarchie](#) est créée lors d'une opération illégale sur la hiérarchie de comptes.

```
#include <ExceptionCompte.h>
```

Est dérivée de [Exception](#).

Types publics

— enum [CodeExcepHierarchie](#) {
[UKWN_EXC_H](#) = 0, [TYPE_EXC_H](#) = 1, [POSTE_EXC_H](#) = 2, [RULE_EXC_H](#) = 3,
[IDNF_EXC_H](#) = 4, [FLSR_EXC_H](#) = 5 }

Fonctions membres publiques

— [ExceptionHierarchie](#) (string _s, [CodeExcepHierarchie](#) _code_erreur=[CodeExcepHierarchie::UKWN_EXC↔
_H](#)) noexcept
Constructeur à partir d'un message d'erreur.

Membres hérités additionnels

4.10.1 Description détaillée

[ExceptionHierarchie](#) est créée lors d'une opération illégale sur la hiérarchie de comptes.

4.10.2 Documentation des énumérations membres

4.10.2.1 CodeExcepHierarchie

```
enum ExceptionHierarchie::CodeExcepHierarchie
```

Valeurs énumérées

UKWN_EXC_H	0 : une exception inconnue
TYPE_EXC_H	1 : Un type de Compte inconnue
POSTE_EXC_H	2 : Un compte de Poste inconnu
RULE_EXC_H	3 : Ne respecte par les règles de création d'un compte : un père virtuel et doit être de même poste que le père
IDNF_EXC_H	4 : Tentative d'accès à un compte non présent dans la hiérarchie.
FLSR_EXC_H	5 : Tentative d'accéder ou manipuler les fils d'un compte réel

4.10.3 Documentation des constructeurs et destructeur

4.10.3.1 ExceptionHierarchie()

```
ExceptionHierarchie::ExceptionHierarchie (
    string _s,
    CodeExcepHierarchie _code_erreur = CodeExcepHierarchie::UKWN\_EXC\_H ) [inline],
[noexcept]
```

Constructeur à partir d'un message d'erreur.

La documentation de cette classe a été générée à partir du fichier suivant :

— [ExceptionCompte.h](#)

4.11 Référence de la classe ExceptionTransaction

traite les exceptions de la partie transaction

```
#include <exceptiontransaction.h>
```

Est dérivée de [Exception](#).

Types publics

```
— enum CodeExcepTransaction {
    UKWN_EXC_T = 40, MEMO_EXC_T = 41, RULE_EXC_T = 42, NULL_EXC_T = 43,
    SRCH_EXC_T = 44 }
```

Fonctions membres publiques

```
— ExceptionTransaction (string _message, CodeExcepTransaction _code_erreur=CodeExcepTransaction::UKWN\_EXC\_T) noexcept
    Constructeur, hérite de Exception.
```

Membres hérités additionnels

4.11.1 Description détaillée

traite les exceptions de la partie transaction

4.11.2 Documentation des énumérations membres

4.11.2.1 [CodeExcepTransaction](#)

```
enum ExceptionTransaction::CodeExcepTransaction
```

Valeurs énumérées

UKWN_EXC_T	40 : une exception inconnue
MEMO_EXC_T	41 : exception levée pour un problème d'allocation mémoire
RULE_EXC_T	42 : exception levée lorsqu'un résultat ne correspond pas aux règles imposées par la logique de l'application
NULL_EXC_T	43 : exception levée lorsqu'un pointeur NULLPTR est non-attendu
SRCH_EXC_T	44 : exception levée lorsqu'une boucle de recherche ne trouve aucun l'élément

4.11.3 Documentation des constructeurs et destructeur

4.11.3.1 [ExceptionTransaction\(\)](#)

```
ExceptionTransaction::ExceptionTransaction (
    string _message,
```

```
CodeExceptTransaction _code_erreur = CodeExceptTransaction::UKWN_EXC_T ) [inline],
[explicit], [noexcept]
```

Constructeur, hérite de [Exception](#).

Paramètres

<code>_message</code>	: string, message d'erreur expliquant l'exception
<code>_code_erreur</code>	: CodeExceptTransaction, code d'erreur permettant le traitement de l'exception

La documentation de cette classe a été générée à partir du fichier suivant :
— [exceptiontransaction.h](#)

4.12 Référence de la classe HierarchieBuilder

[HierarchieBuilder](#) gère la création de la hiérarchie des comptes.

```
#include <HierarchieBuilder.h>
```

Fonctions membres privées statiques

- static [CompteHierarchie](#) * [creerEmpty](#) () noexcept
Méthode qui permet d'initialiser une hiérarchie de comptes vide : crée le compte racine.
- static [CompteHierarchie](#) * [creerFils](#) (string _nom, [Type](#) _t, [Poste](#) _p)
Méthode qui permet de créer un nouveau compte.
- static [CompteHierarchie](#) * [creerAvecFichier](#) (string _path)
Méthode qui permet de créer une hiérarchie à partir d'un fichier.

Attributs privés statiques

- static int [idcompte](#) = 0

Amis

- class [CompteManager](#)

4.12.1 Description détaillée

[HierarchieBuilder](#) gère la création de la hiérarchie des comptes.

4.12.2 Documentation des fonctions membres

4.12.2.1 [creerAvecFichier\(\)](#)

```
CompteHierarchie * HierarchieBuilder::creerAvecFichier (
    string _path ) [static], [private]
```

Méthode qui permet de créer une hiérarchie à partir d'un fichier.

Paramètres

<code>_path</code>	: string, chemin vers le fichier à utiliser.
--------------------	--

Exceptions

<i>non</i>	implémenté
------------	------------

Renvoie

Un pointeur vers l'élément racine de la hiérarchie construite.

4.12.2.2 `creerEmpty()`

```
CompteHierarchie * HierarchieBuilder::creerEmpty ( ) [static], [private], [noexcept]
```

Méthode qui permet d'initialiser une hiérarchie de comptes vide : crée le compte racine.

Renvoie

Un pointeur vers l'élément racine d'une hiérarchie vide.

4.12.2.3 `creerFils()`

```
CompteHierarchie * HierarchieBuilder::creerFils (
    string _nom,
    Type _t,
    Poste _p ) [static], [private]
```

Méthode qui permet de créer un nouveau compte.

Paramètres

<code>_nom</code>	: string, nom du nouveau compte
<code>_t</code>	: Type, Type du nouveau compte : VIRTUEL ou REEL
<code>_p</code>	: Poste, Poste du nouveau compte : DEPENSE ou RECETTE ou ACTIF ou PASSIF

Exceptions

<code>TYPE_EXC_H</code>	Type de compte inconnu
<code>POSTE_EXC_↔_H</code>	Type de poste inconnu

Renvoie

Un pointeur vers un nouvel élément [CompteHierarchie](#).

4.12.3 Documentation des fonctions amies et associées

4.12.3.1 CompteManager

```
friend class CompteManager [friend]
```

4.12.4 Documentation des données membres

4.12.4.1 idcompte

```
int HierarchieBuilder::idcompte = 0 [static], [private]
```

La documentation de cette classe a été générée à partir du fichier suivant :

- [HierarchieBuilder.h](#)
- [HierarchieBuilder.cpp](#)

4.13 Référence de la classe Memento

Le [Memento](#) permet de sauvegarder la date et le solde du dernier rapprochement.

```
#include <Memento.h>
```

Fonctions membres publiques

- [Memento](#) (const int __solde) noexcept
Constructeur.
- double [getSolde](#) () const noexcept
Méthode qui retourne le solde du memento.
- time_t [getDate](#) () const noexcept
Méthode qui permet d'obtenir la date du memento.

Attributs privés

- time_t [date](#)
Correspond à la date où l'on enregistre le solde dans le memento.
- double [solde](#)
Solde du compte à la date date.

4.13.1 Description détaillée

Le [Memento](#) permet de sauvegarder la date et le solde du dernier rapprochement.

4.13.2 Documentation des constructeurs et destructeur

4.13.2.1 Memento()

```
Memento::Memento (
    const int _solde ) [noexcept]
```

Constructeur.

4.13.3 Documentation des fonctions membres

4.13.3.1 getDate()

```
time_t Memento::getDate ( ) const [inline], [noexcept]
```

Méthode qui permet d'obtenir la date du memento.

Renvoie

La date où a été enregistré le solde dans le memento.

4.13.3.2 getSolde()

```
int Memento::getSolde ( ) const [inline], [noexcept]
```

Méthode qui retourne le solde du memento.

Renvoie

Un double qui représente le solde enregistré.

4.13.4 Documentation des données membres

4.13.4.1 date

```
time_t Memento::date [private]
```

Correspond à la date où l'on enregistre le solde dans le memento.

4.13.4.2 solde

```
int Memento::solde [private]
```

Solde du compte à la date date.

La documentation de cette classe a été générée à partir du fichier suivant :

- [Memento.h](#)
- [Memento.cpp](#)

4.14 Référence de la classe Operation

Objet représentant une opération comptable.

```
#include <operation.h>
```

Fonctions membres publiques

- [~Operation](#) ()
Destructeur.
- bool [isCompte](#) (int const _id_compte) const noexcept
vérifier si l'opération appartient au compte correspondant
- int [getIdCompte](#) () const noexcept
Obtenir l'id du compte de l'opération.
- double const [getDebit](#) () const noexcept
obtenir la valeur du débit de l'opération
- double const [getCredit](#) () const noexcept
obtenir la valeur du crédit de l'opération
- bool [operator<](#) ([Operation](#) &_operation) const noexcept
définition de l'opérateur <, test sur crédit et débit
- bool [operator<=](#) ([Operation](#) &_operation) const noexcept
définition de l'opérateur <=, test sur crédit et débit
- bool [operator>](#) ([Operation](#) &_operation) const noexcept
définition de l'opérateur >, test sur crédit et débit
- bool [operator>=](#) ([Operation](#) &_operation) const noexcept
définition de l'opérateur >=, test sur crédit et débit

Fonctions membres privées

- [Operation](#) ([CompteHierarchie](#) *_compte, double _debit, double _credit)
Constructeur, si ni le débit, ni le crédit n'est nul, une soustraction du plus faible montant sur le plus élevé a lieu.

Attributs privés

- friend [TransactionBuilder](#)
- [Transaction](#) * [transaction](#)
la transaction dont l'opération fait parti
- [CompteHierarchie](#) * [compte](#)
le compte sur lequel l'opération porte
- double [debit](#)
la valeur de débit
- double [credit](#)

Amis

- `list< Operation * > operator<< (list< Operation * > &_listOperations, Operation *_operation) noexcept`

4.14.1 Description détaillée

Objet représentant une opération comptable.

4.14.2 Documentation des constructeurs et destructeur

4.14.2.1 ~Operation()

```
Operation::~Operation ( )
```

Destructeur.

4.14.2.2 Operation()

```
Operation::Operation (
    CompteHierarchie * _compte,
    double _debit,
    double _credit ) [explicit], [private]
```

Constructeur, si ni le débit, ni le crédit n'est nul, une soustraction du plus faible montant sur le plus élevé a lieu.

Paramètres

<code>_compte</code>	: CompteHierarchie *, compte sur lequel porte l'opération
<code>_debit</code>	: double, valeur du débit
<code>_credit</code>	: double, valeur du crédit

Exceptions

<i>RULE_EXC↔ _T</i>	le débit et le crédit sont tous les deux nuls
<i>RULE_EXC↔ _T</i>	passage d'un compte non reel
<i>NULL_EXC↔ _T</i>	passage d'un compte nul

4.14.3 Documentation des fonctions membres

4.14.3.1 getCredit()

```
double const Operation::getCredit ( ) const [noexcept]
```

obtenir la valeur du crédit de l'opération

Renvoie

la valeur du crédit de l'opération

4.14.3.2 getDebit()

```
double const Operation::getDebit ( ) const [noexcept]
```

obtenir la valeur du débit de l'opération

Renvoie

la valeur du débit de l'opération

4.14.3.3 getIdCompte()

```
int Operation::getIdCompte ( ) const [noexcept]
```

Obtenir l'id du compte de l'opération.

Renvoie

l'id du compte de l'opération

4.14.3.4 isCompte()

```
bool Operation::isCompte (
    int const _id_compte ) const [noexcept]
```

vérifier si l'opération appartient au compte correspondant

Paramètres

<code>_id_compte</code>	: int const, identifiant d'un compte
-------------------------	--------------------------------------

Renvoie

true si le compte correspond, false sinon

4.14.3.5 operator<()

```
bool Operation::operator< (
    Operation & _operation ) const [noexcept]
```

définition de l'opérateur <, test sur crédit et débit

4.14.3.6 operator<=()

```
bool Operation::operator<= (
    Operation & _operation ) const [noexcept]
```

définition de l'opérateur <=, test sur crédit et débit

4.14.3.7 operator>()

```
bool Operation::operator> (
    Operation & _operation ) const [noexcept]
```

définition de l'opérateur >, test sur crédit et débit

4.14.3.8 operator>=()

```
bool Operation::operator>= (
    Operation & _operation ) const [noexcept]
```

définition de l'opérateur >=, test sur crédit et débit

4.14.4 Documentation des fonctions amies et associées

4.14.4.1 operator<<

```
list<Operation*> operator<< (
    list< Operation * > & _listOperations,
    Operation * _operation ) [friend]
```

4.14.5 Documentation des données membres

4.14.5.1 compte

```
CompteHierarchie * Operation::compte [private]
```

le compte sur lequel l'opération porte

4.14.5.2 credit

```
double Operation::credit [private]
```

4.14.5.3 debit

```
double Operation::debit [private]
```

la valeur de débit

4.14.5.4 transaction

```
Transaction * Operation::transaction [private]
```

la transaction dont l'opération fait parti

4.14.5.5 TransactionBuilder

```
friend Operation::TransactionBuilder [private]
```

La documentation de cette classe a été générée à partir du fichier suivant :

- [operation.h](#)
- [operation.cpp](#)

4.15 Référence de la classe Originator

[Originator](#) permet de manipuler les mementos.

```
#include <Originator.h>
```

Fonctions membres publiques

- [Originator](#) ()=default
Constructeur.
- void [setSolde](#) (const int _solde) noexcept
Méthode qui permet de modifier le solde.
- double [getSolde](#) () noexcept
Méthode qui retourne le solde.
- const time_t & [getDate](#) () noexcept
Méthode qui retourne le data où le solde a été enregistré.
- [Memento](#) * [saveState](#) () noexcept
Méthode qui permet d'enregistrer le solde dans une nouveau [Memento](#).
- void [restoreMemento](#) ([Memento](#) _memento) noexcept

Attributs privés

- time_t [date](#)
Correspond à la date où l'on enregistre ou modifie le solde.
- double [solde](#)
Solde du compte à la date date.

4.15.1 Description détaillée

[Originator](#) permet de manipuler les mementos.

4.15.2 Documentation des constructeurs et destructeur

4.15.2.1 Originator()

```
Originator::Originator ( ) [default]
```

Constructeur.

4.15.3 Documentation des fonctions membres

4.15.3.1 getDate()

```
const time_t & Originator::getDate ( ) [inline], [noexcept]
```

Méthode qui retourne le data où le solde a été enregistré.

Renvoie

Date d'enregistrement du solde.

4.15.3.2 getSolde()

```
int Originator::getSolde ( ) [inline], [noexcept]
```

Méthode qui retourne le solde.

Renvoie

Un double qui représente le solde.

4.15.3.3 restoreMemento()

```
void Originator::restoreMemento (
    Memento _memento ) [inline], [noexcept]
```

4.15.3.4 saveState()

```
Memento * Originator::saveState ( ) [inline], [noexcept]
```

Méthode qui permet d'enregistrer le solde dans un nouveau [Memento](#).

Renvoie

Un pointeur sur le [Memento](#) créé.

4.15.3.5 setSolde()

```
void Originator::setSolde (
    const int _solde ) [inline], [noexcept]
```

Méthode qui permet de modifier le solde.

Paramètres

<code>_solde</code>	: const int, nouveau solde
---------------------	----------------------------

4.15.4 Documentation des données membres

4.15.4.1 date

```
time_t Originator::date [private]
```

Correspond à la date où l'on enregistre ou modifie le solde.

4.15.4.2 solde

```
double Originator::solde [private]
```

Solde du compte à la date date.

La documentation de cette classe a été générée à partir du fichier suivant :

— [Originator.h](#)

4.16 Référence de la classe Passif

[Passif](#) permet de représenter les propriétés des comptes de poste [Passif](#).

```
#include <Passif.h>
```

Est dérivée de [PostePR](#).

Fonctions membres publiques

- [Poste](#) `getPoste ()` const noexcept override
Redéfinition de la méthode virtuelle `getPoste` de [PosteCompte](#). Cette méthode retourne le nom du poste passif : PASSIF.
- `string` `getLetter ()` const noexcept override
Redéfinition de la méthode virtuelle `getLetter` de [PosteCompte](#). Cette méthode renvoie la première lettre de le nom d'un poste passif : P.

Fonctions membres publiques statiques

- static const [PosteCompte](#) & `getInstance ()` noexcept
Méthode qui permet d'obtenir l'instance [Passif](#).

Fonctions membres privées

- `Passif()`=default
Constructeur privé
- `~Passif()`=default
Destructeur privé
- `Passif(const Passif &_p)`=default
Constructeur par recopie privé
- `Passif & operator= (const Passif &_p)`=default
Surcharge de l'opérateur d'affectation privé

Attributs privés statiques

- static `Passif instance`
instance `Passif`

4.16.1 Description détaillée

`Passif` permet de représenter les propriétés des comptes de poste `Passif`.

4.16.2 Documentation des constructeurs et destructeur

4.16.2.1 `Passif()` [1/2]

```
Passif::Passif ( ) [private], [default]
```

Constructeur privé

4.16.2.2 `~Passif()`

```
Passif::~~Passif ( ) [private], [default]
```

Destructeur privé

4.16.2.3 `Passif()` [2/2]

```
Passif::Passif (
    const Passif & _p ) [private], [default]
```

Constructeur par recopie privé

Paramètres

↔	: const Passif &, référence constante sur le poste passif à recopier.
↔	
<i>p</i>	

4.16.3 Documentation des fonctions membres

4.16.3.1 getInstance()

```
static Passif & Passif::getInstance ( ) [inline], [static], [noexcept]
```

Méthode qui permet d'obtenir l'instance [Passif](#).

Renvoie

instance

4.16.3.2 getLetter()

```
string Passif::getLetter ( ) const [inline], [override], [virtual], [noexcept]
```

Redéfinition de la méthode virtuelle getLetter de [PosteCompte](#). Cette méthode renvoie la première lettre de le nom d'un poste passif : P.

Renvoie

P

Implémente [PosteCompte](#).

4.16.3.3 getPoste()

```
Poste Passif::getPoste ( ) const [inline], [override], [virtual], [noexcept]
```

Redéfinition de la méthode virtuelle getPoste de [PosteCompte](#). Cette méthode retourne le nom du poste passif : PASSIF.

Renvoie

PASSIF

Implémente [PosteCompte](#).

4.16.3.4 operator=()

```
const PosteCompte & Passif::operator= (
    const Passif & _p ) [private], [default]
```

Surcharge de l'opérateur d'affectation privé

Paramètres

↔	: const Passif &, référence constante sur le poste passif à affecter.
↔	
<i>p</i>	

4.16.4 Documentation des données membres

4.16.4.1 instance

```
static Passif::Passif Passif::instance [static], [private]
```

instance [Passif](#)

La documentation de cette classe a été générée à partir du fichier suivant :

— [Passif.h](#)
— [Passif.cpp](#)

4.17 Référence de la classe PosteAD

[PosteAD](#) est la classe mère des postes de compte : [Actif](#) et [Depense](#). Elle a pour but de définir les méthodes créditer et débiter qui sont communes à ces deux classes.

```
#include <PosteAD.h>
```

Est dérivée de [PosteCompte](#).

Dérivée par [Actif](#), et [Depense](#).

Fonctions membres publiques

- void [debiter](#) (double *_solde, double _montant) const noexcept override
Redéfinition de la méthode debiter de [PosteCompte](#). Cette méthode permet de débiter de l'argent d'un compte actif ou de dépenses, c'est-à-dire qu'elle augmente le solde d'un de ces comptes.
- void [crediter](#) (double *_solde, double _montant) const noexcept override
Redéfinition de la méthode crediter de [PosteCompte](#). Cette méthode permet de créditer de l'argent sur compte actif ou de dépenses, c'est-à-dire qu'elle diminue le solde d'un de ces comptes.

4.17.1 Description détaillée

[PosteAD](#) est la classe mère des postes de compte : [Actif](#) et [Depense](#). Elle a pour but de définir les méthodes créditer et débiter qui sont communes à ces deux classes.

4.17.2 Documentation des fonctions membres

4.17.2.1 crediter()

```
void PosteAD::crediter (
    double * _solde,
    double _montant ) const [inline], [override], [virtual], [noexcept]
```

Redéfinition de la méthode crediter de [PosteCompte](#). Cette méthode permet de créditer de l'argent sur compte actif ou de dépenses, c'est-à-dire qu'elle diminue le solde d'un de ces comptes.

Paramètres

<code>_solde</code>	: double*, pointeur sur le solde du compte que l'on veut créditer.
<code>_montant</code>	: double, montant que l'on veut créditer sur compte.

Implémente [PosteCompte](#).

4.17.2.2 debiter()

```
void PosteAD::debiter (
    double * _solde,
    double _montant ) const [inline], [override], [virtual], [noexcept]
```

Redéfinition de la méthode debiter de [PosteCompte](#). Cette méthode permet de débiter de l'argent d'un compte actif ou de dépenses, c'est-à-dire qu'elle augmente le solde d'un de ces comptes.

Paramètres

<code>_solde</code>	: double*, pointeur sur le solde du compte que l'on veut débiter.
<code>_montant</code>	: double, montant que l'on veut débiter du compte.

Implémente [PosteCompte](#).

La documentation de cette classe a été générée à partir du fichier suivant :

— [PosteAD.h](#)

4.18 Référence de la classe PosteCompte

[PosteCompte](#) est la classe mère des différents postes de compte : [Actif](#) et [Dépense](#), [Passif](#) et [Recette](#), et la [Racine](#).

```
#include <PosteCompte.h>
```

Dérivée par [PosteAD](#), [PostePR](#), et [Racine](#).

Fonctions membres publiques

- virtual string [getLetter](#) () const noexcept=0
Méthode virtuelle qui permettra de renvoyer la première lettre de le nom du poste d'un compte.
- virtual [Poste](#) [getPoste](#) () const noexcept=0
Méthode qui permet de retourner le nom du poste d'un [PosteCompte](#).
- virtual void [debiter](#) (double * _solde, double _montant) const =0
- virtual void [crediter](#) (double * _solde, double _montant) const =0

4.18.1 Description détaillée

[PosteCompte](#) est la classe mère des différents postes de compte : [Actif](#) et [Dépense](#), [Passif](#) et [Recette](#), et la [Racine](#).

4.18.2 Documentation des fonctions membres

4.18.2.1 crediter()

```
virtual void PosteCompte::crediter (
    double * _solde,
    double _montant ) const [pure virtual]
```

Implémenté dans [Racine](#), [PosteAD](#), et [PostePR](#).

4.18.2.2 debiter()

```
virtual void PosteCompte::debiter (
    double * _solde,
    double _montant ) const [pure virtual]
```

Implémenté dans [Racine](#), [PosteAD](#), et [PostePR](#).

4.18.2.3 getLetter()

```
string PosteCompte::getLetter ( ) const [pure virtual], [noexcept]
```

Méthode virtuelle qui permettra de renvoyer la première lettre de le nom du poste d'un compte.

Renvoie

D si c'est un poste [Depense](#), R si c'est poste [Recette](#) ou [Racine](#), P si c'est un poste [Passif](#), A si c'est un poste [Actif](#).

Implémenté dans [Depense](#), [Passif](#), [Racine](#), [Recette](#), et [Actif](#).

4.18.2.4 getPoste()

```
Poste PosteCompte::getPoste ( ) const [pure virtual], [noexcept]
```

Méthode qui permet de retourner le nom du poste d'un [PosteCompte](#).

Renvoie

ACTIF si c'est un [Actif](#), PASSIF si c'est un [Passif](#), RECETTE si c'est un [Recette](#), DEPENSE si c'est un [Depense](#) ou RACINE si c'est la racine.

Implémenté dans [Depense](#), [Passif](#), [Racine](#), [Recette](#), et [Actif](#).

La documentation de cette classe a été générée à partir du fichier suivant :
— [PosteCompte.h](#)

4.19 Référence de la classe PostePR

[PostePR](#) est la classe mère des postes de compte : [Recette](#) et [Passif](#). Elle a pour but de définir les méthodes créditer et débiter qui sont communes à ces deux classes.

```
#include <PostePR.h>
```

Est dérivée de [PosteCompte](#).

Dérivée par [Passif](#), et [Recette](#).

Fonctions membres publiques

- void [debiter](#) (double * _solde, double _montant) const noexcept override
Redéfinition de la méthode debiter de [PosteCompte](#). Cette méthode permet de débiter de l'argent d'un compte passif ou de recette, c'est-à-dire qu'elle diminue le solde d'un de ces comptes.
- void [crediter](#) (double * _solde, double _montant) const noexcept override
Redéfinition de la méthode crediter de [PosteCompte](#). Cette méthode permet de créditer de l'argent sur compte passif ou de recette, c'est-à-dire qu'elle augmente le solde d'un de ces comptes.

4.19.1 Description détaillée

[PostePR](#) est la classe mère des postes de compte : [Recette](#) et [Passif](#). Elle a pour but de définir les méthodes créditer et débiter qui sont communes à ces deux classes.

4.19.2 Documentation des fonctions membres

4.19.2.1 crediter()

```
void PostePR::crediter (
    double * _solde,
    double _montant ) const [inline], [override], [virtual], [noexcept]
```

Redéfinition de la méthode crediter de [PosteCompte](#). Cette méthode permet de créditer de l'argent sur compte passif ou de recette, c'est-à-dire qu'elle augmente le solde d'un de ces comptes.

Paramètres

<code>_solde</code>	: double*, pointeur sur le solde du compte que l'on veut créditer.
<code>_montant</code>	: double, montant que l'on veut créditer sur compte.

Implémente [PosteCompte](#).

4.19.2.2 debiter()

```
void PostePR::debiter (
    double * _solde,
    double _montant ) const [inline], [override], [virtual], [noexcept]
```

Redéfinition de la méthode debiter de [PosteCompte](#). Cette méthode permet de débiter de l'argent d'un compte passif ou de recette, c'est-à-dire qu'elle diminue le solde d'un de ces comptes.

Paramètres

<code>_solde</code>	: double*, pointeur sur le solde du compte que l'on veut débiter.
<code>_montant</code>	: double, montant que l'on veut débiter du compte.

Implémente [PosteCompte](#).

La documentation de cette classe a été générée à partir du fichier suivant :

— [PostePR.h](#)

4.20 Référence de la classe Racine

[Racine](#) permet de représenter le comportement d'un compte [Racine](#), c'est-à-dire celui qui contient tous les autres comptes.

```
#include <Racine.h>
```

Est dérivée de [PosteCompte](#).

Fonctions membres publiques

- [Poste](#) [getPoste](#) () const noexcept override
Redéfinition de la méthode virtuelle [getPoste](#) de [PosteCompte](#). Cette méthode retourne le nom du poste [Racine](#): RACINE.
- string [getLetter](#) () const noexcept override
Réfinition de la méthode virtuelle [getLetter](#) de [PosteCompte](#). Cette méthode renvoie la première lettre d'un poste [Racine](#) : R.
- void [debiter](#) (double * _solde, double _montant) const override
Réfinition de la méthode virtuelle [getPoste](#) de [PosteCompte](#) : lancement d'une exception, on ne peut pas débiter le compte racine.
- void [crediter](#) (double * _solde, double _montant) const override
Réfinition de la méthode virtuelle [getPoste](#) de [PosteCompte](#) : lancement d'une exception, on ne peut pas créditer le compte racine.

Fonctions membres publiques statiques

- static const [PosteCompte](#) & [getInstance](#) () noexcept
Méthode qui permet d'obtenir l'instance [Racine](#).

Fonctions membres privées

- [Racine](#) ()=default
Constructeur privé
- [~Racine](#) ()=default
Destructeur privé
- [Racine](#) (const [Racine](#) &_r)=delete
Suppression constructeur par recopie.
- [Racine](#) & [operator=](#) (const [Racine](#) &_r)=delete

Attributs privés statiques

- static [Racine](#) instance
instance [Racine](#)

4.20.1 Description détaillée

[Racine](#) permet de représenter le comportement d'un compte [Racine](#), c'est-à-dire celui qui contient tous les autres comptes.

4.20.2 Documentation des constructeurs et destructeur

4.20.2.1 [Racine](#)() [1/2]

```
Racine::Racine ( ) [private], [default]
```

Constructeur privé

4.20.2.2 [~Racine](#)()

```
Racine::~~Racine ( ) [private], [default]
```

Destructeur privé

4.20.2.3 [Racine](#)() [2/2]

```
Racine::Racine (
    const Racine &_r ) [private], [delete]
```

Suppression constructeur par recopie.

4.20.3 Documentation des fonctions membres

4.20.3.1 [crediter](#)()

```
void Racine::crediter (
    double * _solde,
    double _montant ) const [override], [virtual]
```

Réfinition de la méthode virtuelle `getPoste` de [PosteCompte](#) : lancement d'une exception, on ne peut pas créditer le compte [racine](#).

Exceptions

<code>ROOT_EXC↔ _C</code>	Lancement d'ExceptionComptabilité : Impossible de créditer le compte racine.
-------------------------------	--

Implémente [PosteCompte](#).

4.20.3.2 debiter()

```
void Racine::debiter (
    double * _solde,
    double _montant ) const [override], [virtual]
```

Réfinition de la méthode virtuelle getPoste de [PosteCompte](#) : lancement d'une exception, on ne peut pas débiter le compte racine.

Exceptions

<code>ROOT_EXC↔ _C</code>	Lancement d'ExceptionComptabilité : Impossible de débiter le compte racine.
-------------------------------	---

Implémente [PosteCompte](#).

4.20.3.3 getInstance()

```
static const PosteCompte & Racine::getInstance ( ) [inline], [static], [noexcept]
```

Méthode qui permet d'obtenir l'instance [Racine](#).

Renvoie

instance

4.20.3.4 getLetter()

```
string Racine::getLetter ( ) const [inline], [override], [virtual], [noexcept]
```

Réfinition de la méthode virtuelle getLetter de [PosteCompte](#). Cette méthode renvoie la première lettre d'un poste [Racine](#) : R.

Renvoie

R

Implémente [PosteCompte](#).

4.20.3.5 `getPoste()`

```
Poste Racine::getPoste ( ) const [inline], [override], [virtual], [noexcept]
```

Redéfinition de la méthode virtuelle `getPoste` de [PosteCompte](#). Cette méthode retourne le nom du poste [Racine](#): RACINE.

Renvoie

RACINE

Implémente [PosteCompte](#).

4.20.3.6 `operator=()`

```
Racine& Racine::operator= (
    const Racine & _r ) [private], [delete]
```

4.20.4 Documentation des données membres

4.20.4.1 instance

```
static Racine::Racine Racine::instance [static], [private]
```

instance [Racine](#)

La documentation de cette classe a été générée à partir du fichier suivant :

- [Racine.h](#)
- [Racine.cpp](#)

4.21 Référence de la classe [Recette](#)

[Recette](#) permet de représenter le comportement des comptes de poste [Recette](#).

```
#include <Recette.h>
```

Est dérivée de [PostePR](#).

Fonctions membres publiques

- [Poste](#) `getPoste ()` const noexcept override
Réfinition de la méthode virtuelle `getPoste` de [PosteCompte](#). Cette méthode retourne le nom du poste [Recette](#): RECETTE.
- string `getLetter ()` const noexcept override
Réfinition de la méthode virtuelle `getLetter` de [PosteCompte](#). Cette méthode renvoie la première lettre du nom d'un poste [Recette](#) : R.

Fonctions membres publiques statiques

- static const PosteCompte & getInstance () noexcept
Méthode qui permet d'obtenir l'instance Recette.

Fonctions membres privées

- Recette ()=default
Constructeur privé
- ~Recette ()=default
Destructeur privé
- Recette (const Recette &_r)=default
Constructeur par copie privé
- Recette & operator= (const Recette &_r)=default
Surcharge de l'opérateur d'affectation privé

Attributs privés statiques

- static Recette instance
instance Recette

4.21.1 Description détaillée

Recette permet de représenter le comportement des comptes de poste Recette.

4.21.2 Documentation des constructeurs et destructeur

4.21.2.1 Recette() [1/2]

```
Recette::Recette ( ) [private], [default]
```

Constructeur privé

4.21.2.2 ~Recette()

```
Recette::~~Recette ( ) [private], [default]
```

Destructeur privé

4.21.2.3 Recette() [2/2]

```
Recette::Recette (
    const Recette & _r ) [private], [default]
```

Constructeur par copie privé

Paramètres

↩	: const Recette &, référence constante sur le poste recette à affecter.
↩	
↩	
r	

4.21.4 Documentation des données membres

4.21.4.1 instance

```
static Recette::Recette Recette::instance [static], [private]
```

instance [Recette](#)

La documentation de cette classe a été générée à partir du fichier suivant :

- [Recette.h](#)
- [Recette.cpp](#)

4.22 Référence de la classe Transaction

La transaction représente un ensemble d'opérations équilibrées effectuées à une même date entre plusieurs comptes.

```
#include <transaction.h>
```

Fonctions membres publiques

- `time_t` [getDate](#) () const noexcept
obtenir la date de la transaction
- `string` [getStringDate](#) () const noexcept
obtenir la date de la transaction au format string
- `string` [getReference](#) () const noexcept
obtenir la référence de la transaction
- `string` [getMemo](#) () const noexcept
obtenir le memo de la transaction
- `bool` [getValide](#) () const noexcept
vérifier si la transaction est rapprochée
- `list< Operation * >` const [getListeOperations](#) () const noexcept
obtenir la liste des opérations de la transaction
- `bool` [operator<](#) ([Transaction](#) &_transaction) const noexcept
définition de l'opérateur <, test sur la date
- `bool` [operator<=](#) ([Transaction](#) &_transaction) const noexcept
définition de l'opérateur <=, test sur la date
- `bool` [operator>](#) ([Transaction](#) &_transaction) const noexcept
définition de l'opérateur >, test sur la date
- `bool` [operator>=](#) ([Transaction](#) &_transaction) const noexcept
définition de l'opérateur >=, test sur la date

Fonctions membres privées

- `Transaction()`=default
Constructeur par défaut.
- `Transaction` (time_t _date, string _ref, string _memo, bool _valide, list< `Operation` * > _listOperations)
Constructeur.
- `~Transaction()`
Destructeur.
- void `setValide` (bool _valide) noexcept
modifieur de l'attribut valide

Attributs privés

- friend `TransactionBuilder`
- friend `CompteManager`
- time_t `date` = 0
la date de la transaction
- string `reference_transaction` = ""
la reference unique de la transaction
- string `memo` = ""
le memo de la transaction
- bool `valide` = false
le marqueur de rapprochement de la transaction. Vaut true si rapprochée, sinon false
- list< `Operation` * > `operations`
la liste des operations composant la transaction

4.22.1 Description détaillée

La transaction représente un ensemble d'opérations équilibrées effectuées à une même date entre plusieurs comptes.

4.22.2 Documentation des constructeurs et destructeur

4.22.2.1 Transaction() [1/2]

```
Transaction::Transaction ( ) [private], [default]
```

Constructeur par défaut.

Destructeur.

4.22.2.2 Transaction() [2/2]

```
Transaction::Transaction (
    time_t _date,
    string _ref,
    string _memo,
    bool _valide,
    list< Operation * > _listOperations ) [explicit], [private]
```

Constructeur.

Paramètres

<code>_date</code>	: <code>time_t</code> , la nouvelle date de la transaction
<code>_ref</code>	: <code>string</code> , la nouvelle référence de la transaction
<code>_memo</code>	: <code>string</code> , le nouveau mémo de la transaction
<code>_valide</code>	: <code>bool</code> , marqueur de rapprochement de la transaction
<code>_listOperations</code>	: <code>list<Operation*></code> , le nouvel ensemble des opérations formant la transaction

Exceptions

<code>RULE_EXC↔ _T</code>	non-équilibre des opérations
-------------------------------	------------------------------

4.22.2.3 ~Transaction()

```
Transaction::~~Transaction ( ) [private]
```

4.22.3 Documentation des fonctions membres

4.22.3.1 getDate()

```
time_t Transaction::getDate ( ) const [noexcept]
```

obtenir la date de la transaction

Renvoie

la date de la transaction

4.22.3.2 getListeOperations()

```
list< Operation * > const Transaction::getListeOperations ( ) const [noexcept]
```

obtenir la liste des opérations de la transaction

Renvoie

la liste des opérations de la transaction

4.22.3.3 `getMemo()`

```
string Transaction::getMemo ( ) const [noexcept]
```

obtenir le memo de la transaction

Renvoie

le memo de la transaction

4.22.3.4 `getReference()`

```
string Transaction::getReference ( ) const [noexcept]
```

obtenir la référence de la transaction

Renvoie

la référence de la transaction

4.22.3.5 `getStringDate()`

```
string Transaction::getStringDate ( ) const [noexcept]
```

obtenir la date de la transaction au format string

Renvoie

la date de la transaction au format jj/mm/aaaa

4.22.3.6 `getValide()`

```
bool Transaction::getValide ( ) const [noexcept]
```

vérifier si la transaction est rapprochée

Renvoie

true si la transaction est rapprochée, sinon false

4.22.3.7 operator<()

```
Transaction::operator< (
    Transaction & _transaction ) const    [noexcept]
```

définition de l'opérateur <, test sur la date

4.22.3.8 operator<=()

```
Transaction::operator<= (
    Transaction & _transaction ) const    [noexcept]
```

définition de l'opérateur <=, test sur la date

4.22.3.9 operator>()

```
Transaction::operator> (
    Transaction & _transaction ) const    [noexcept]
```

définition de l'opérateur >, test sur la date

4.22.3.10 operator>=()

```
Transaction::operator>= (
    Transaction & _transaction ) const    [noexcept]
```

définition de l'opérateur >=, test sur la date

4.22.3.11 setValide()

```
Transaction::setValide (
    bool _valide )    [private], [noexcept]
```

modifieur de l'attribut valide

Paramètres

<code>_valide</code>	: bool, nouvelle valeur de l'attribut valide
----------------------	--

4.22.4 Documentation des données membres

4.22.4.1 CompteManager

```
friend Transaction::CompteManager [private]
```

4.22.4.2 date

```
time_t Transaction::date = 0 [private]
```

la date de la transaction

4.22.4.3 memo

```
string Transaction::memo = "" [private]
```

le memo de la transaction

4.22.4.4 operations

```
list< Operation * > Transaction::operations [private]
```

la liste des operations composant la transaction

4.22.4.5 reference_transaction

```
string Transaction::reference_transaction = "" [private]
```

la reference unique de la transaction

4.22.4.6 TransactionBuilder

```
friend Transaction::TransactionBuilder [private]
```

4.22.4.7 valide

```
bool Transaction::valide = false [private]
```

le marqueur de rapprochement de la transaction. Vaut true si rapprochée, sinon false

La documentation de cette classe a été générée à partir du fichier suivant :

- [transaction.h](#)
- [transaction.cpp](#)

4.23 Référence de la classe TransactionBuilder

Le transaction builder permet de construire les transactions. Il utilise le pattern builder et singleton.

```
#include <transactionbuilder.h>
```

Fonctions membres privées

- [TransactionBuilder](#) ()
Constructeur, en tant que singleton ce dernier est privé
- [~TransactionBuilder](#) ()
Destructeur.

Fonctions membres privées statiques

- static void [destruireTransaction](#) ([Transaction](#) *_transaction) noexcept
détruit la liste de transaction passée en paramètre
- static [Transaction](#) * [creerTransaction](#) (time_t _date, string _ref, string _memo, list< [Operation](#) * > _list←
Operations)
permet d'ajouter une nouvelle transaction
- static list< [Transaction](#) * > [creerTransactionAvecFichier](#) (string const _path)
permet de convertir la sauvegarde en une liste de transactions
- static [Operation](#) * [creerOperation](#) ([CompteHierarchie](#) *_compte, double _debit, double _credit)
permet d'ajouter une nouvelle opération

Attributs privés

- friend [TransactionManager](#)

Attributs privés statiques

- static [TransactionBuilder](#) * [instance_builder](#) = new [TransactionBuilder](#)()
l'instance du singleton TransactionBuilder

4.23.1 Description détaillée

Le transaction builder permet de construire les transactions. Il utilise le pattern builder et singleton.

4.23.2 Documentation des constructeurs et destructeur

4.23.2.1 TransactionBuilder()

```
TransactionBuilder::TransactionBuilder ( ) [private]
```

Constructeur, en tant que singleton ce dernier est privé

4.23.2.2 ~TransactionBuilder()

```
TransactionBuilder::~~TransactionBuilder ( ) [private]
```

Destructeur.

4.23.3 Documentation des fonctions membres

4.23.3.1 creerOperation()

```
Operation * TransactionBuilder::creerOperation (
    CompteHierarchie * _compte,
    double _debit,
    double _credit ) [static], [private]
```

permet d'ajouter une nouvelle opération

Paramètres

<i>_compte</i>	: CompteHierarchie*, pointeur sur le compte correspondant qui doit être réel
<i>_debit</i>	: double, correspond au montant à débiter
<i>_credit</i>	: double, correspondant au montant à créditer

Exceptions

<i>NULL_EXCEPT_C</i>	exception d'opération : passage d'un compte nul
<i>RULE_EXCEPT_T</i>	exception d'opération : crédit et débit sont tous les deux nuls
<i>RULE_EXCEPT_T</i>	passage d'un compte non réel

Renvoie

retourne un pointeur sur l'opération nouvellement créée

4.23.3.2 creerTransaction()

```
Transaction * TransactionBuilder::creerTransaction (
    time_t _date,
    string _ref,
    string _memo,
    list< Operation * > _listOperations ) [static], [private]
```

permet d'ajouter une nouvelle transaction

Paramètres

<i>_date</i>	: time_t, la date de la transaction à créer
<i>_ref</i>	: string, la référence de la transaction à créer
<i>_memo</i>	: string, le mémoire de la transaction à créer
<i>_listOperations</i>	: list<Operation*>, ensemble des opérations formant la transaction à créer

Exceptions

<i>MEMO_EXC_T</i>	exception d'allocation mémoire
<i>RULE_EXC_T</i>	non-équilibre des opérations
<i>RULE_EXC_T</i>	liste vide d'opérations

Renvoie

retourne un pointeur sur la transaction nouvellement créée

4.23.3.3 creerTransactionAvecFichier()

```
list< Transaction * > TransactionBuilder::creerTransactionAvecFichier (
    string const _path ) [static], [private]
```

permet de convertir la sauvegarde en une liste de transactions

Paramètres

<i>_path</i>	: string const, le chemin vers le fichier de sauvegarde contenant les transactions
--------------	--

Exceptions

<i>UNFD_EXC_F</i>	exception d'accès au fichier
<i>SNTX_EXC_F</i>	exception de syntaxe du fichier
<i>RULE_EXC_T</i>	exception d'opération : le débit et le crédit sont tous les deux nuls
<i>RULE_EXC_T</i>	exception d'opération : non-équilibre des opérations
<i>NULL_EXC_T</i>	exception d'opération : passage d'un compte nul
<i>MEMO_EXC_T</i>	exception d'allocation mémoire

Renvoie

retourne une liste de pointeurs sur les transactions

4.23.3.4 détruireTransaction()

```
void TransactionBuilder::détruireTransaction (
    Transaction * _transaction ) [static], [private], [noexcept]
```

détruit la liste de transaction passée en paramètre

Paramètres

<i>_transaction</i>	: Transaction*, transaction à détruire
---------------------	--

4.23.4 Documentation des données membres

4.23.4.1 instance_builder

```
TransactionBuilder * TransactionBuilder::instance_builder = new TransactionBuilder() [static],
[private]
```

l'instance du singleton [TransactionBuilder](#)

4.23.4.2 TransactionManager

```
friend TransactionBuilder::TransactionManager [private]
```

La documentation de cette classe a été générée à partir du fichier suivant :

- [transactionbuilder.h](#)
- [transactionbuilder.cpp](#)

4.24 Référence de la classe TransactionManager

Le transaction manager permet de gérer les transactions. Elle est basée sur les pattern Manager et Singleton.

```
#include <transactionmanager.h>
```

Fonctions membres publiques

- list< Transaction * > [getListeTransactions](#) () const noexcept
permet d'obtenir la liste de toutes les transactions
- Transaction * [getTransaction](#) (string const _ref) noexcept
permet d'accéder à une transaction depuis sa référence
- list< Transaction * > [getListeTransactionsParCompte](#) (int const _id_compte)
permet d'obtenir une liste des transactions correspondantes à un même compte
- list< Transaction * > [getListeTransactionsParValide](#) (int const _id_compte, bool const _valide)
permet d'obtenir une liste des transactions correspondantes à un même compte
- void [ajouterOperation](#) (list< Operation * > *_liste, CompteHierarchie *_compte, double _debit, double _credit)
permet d'ajouter une nouvelle opération à une liste d'opérations
- void [ajouterTransaction](#) (time_t _date, string _ref, string _memo, list< Operation * > _listOperations)
permet d'ajouter une nouvelle transaction
- void [editerTransaction](#) (string const _ancienne_ref, time_t _date, string _ref, string _memo, list< Operation * > _listOperations)
permet d'éditer une transaction existante
- void [supprimerTransaction](#) (string const _ref)
permet de supprimer une transaction
- void [charger](#) (string const _path)
permet de charger un fichier
- void [sauver](#) (string const _path)
permet de sauvegarder un fichier

Fonctions membres publiques statiques

- static TransactionManager & [getInstance](#) () noexcept
permet d'accéder à l'instance du singleton

Fonctions membres privées

- TransactionManager ()
Constructeur, en tant que singleton ce dernier est privé
- ~TransactionManager ()
Destructeur.

Attributs privés

- string [path](#)
chemin vers le fichier de sauvegarde utilisé

Attributs privés statiques

- static TransactionManager [instance](#)
l'instance du manager
- static list< Transaction * > [transactions](#)
ensemble des transactions existantes

4.24.1 Description détaillée

Le transaction manager permet de gérer les transactions. Elle est basée sur les pattern Manager et Singleton.

4.24.2 Documentation des constructeurs et destructeur

4.24.2.1 TransactionManager()

```
TransactionManager::TransactionManager ( ) [private]
```

Constructeur, en tant que singleton ce dernier est privé

4.24.2.2 ~TransactionManager()

```
TransactionManager::~~TransactionManager ( ) [private]
```

Destructeur.

4.24.3 Documentation des fonctions membres

4.24.3.1 ajouterOperation()

```
void TransactionManager::ajouterOperation (
    list< Operation * > * _liste,
    CompteHierarchie * _compte,
    double _debit,
    double _credit )
```

permet d'ajouter une nouvelle opération à une liste d'opération

Paramètres

<code>_liste</code>	: list<Operation*>*, pointeur sur la liste d'opération à compléter
<code>_compte</code>	: CompteHierarchie*, pointeur sur le compte subissant l'opération
<code>_debit</code>	: double, correspond au montant à débiter
<code>_credit</code>	: double, correspondant au montant à créditer

Exceptions

<i>NULL_EXC↔ _T</i>	exception d'opération : passage d'un compte nul
<i>RULE_EXC↔ _T</i>	exception d'opération : crédit et débit sont tous les deux nuls
<i>RULE_EXC↔ _T</i>	passage d'un compte non reel

4.24.3.2 ajouterTransaction()

```
void TransactionManager::ajouterTransaction (
    time_t _date,
    string _ref,
    string _memo,
    list< Operation * > _listOperations )
```

permet d'ajouter une nouvelle transaction

Paramètres

<i>_date</i>	: time_t, la date de la nouvelle transaction
<i>_ref</i>	: string, la référence de la nouvelle transaction
<i>_memo</i>	: string, le mémoire de la nouvelle transaction
<i>_listOperations</i>	: list<Operation*>, ensemble des opérations formant la nouvelle transaction

Exceptions

<i>RULE_EXC_T</i>	non-unicité de la référence _ref
<i>RULE_EXC_T</i>	non-équilibre des opérations
<i>RULE_EXC_T</i>	tentative d'ajout d'une transaction antérieure au dernier rapprochement
<i>IDNF_EXC_H</i>	l'identifiant du compte lié à une opération ne correspond à aucun compte réel
<i>MEMO_EXC↔ _T</i>	exception d'allocation mémoire

4.24.3.3 charger()

```
void TransactionManager::charger (
    string const _path )
```

permet de charger un fichier

Paramètres

<code>_path</code>	: string const, le chemin du fichier de sauvegarde
--------------------	--

Exceptions

<code>RULE_EXC_T</code>	exception d'opération : le débit et le crédit sont tous les deux nuls
<code>RULE_EXC_T</code>	exception d'opération : non-équilibre des opérations
<code>NULL_EXC_T</code>	exception d'opération : passage d'un compte nul
<code>MEMO_EXC_T</code>	exception d'allocation mémoire

4.24.3.4 `editerTransaction()`

```
void TransactionManager::editerTransaction (
    string const _ancienne_ref,
    time_t _date,
    string _ref,
    string _memo,
    list< Operation * > _listOperations )
```

permet d'editer une transaction existante

Paramètres

<code>_ancienne_ref</code>	: string const, la référence de la transaction à modifier
<code>_date</code>	: time_t, la nouvelle date de la transaction
<code>_ref</code>	: string, la nouvelle référence de la transaction
<code>_memo</code>	: string, le nouveau mémoire de la transaction
<code>_listOperations</code>	: list<Operation*>, le nouvel ensemble des opérations formant la transaction

Exceptions

<code>RULE_EXC_T</code>	non-équilibre des opérations
<code>RULE_EXC_T</code>	non-unicité de la référence <code>_ref</code>
<code>RULE_EXC_T</code>	tentative de modification sur une référence rapprochée
<code>MEMO_EXC_T</code>	exception d'allocation mémoire
<code>SRCH_EXC_T</code>	référence <code>_ancienne_ref</code> inconnue

4.24.3.5 getInstance()

```
TransactionManager & TransactionManager::getInstance ( ) [static], [noexcept]
```

permet d'accéder à l'instance du singleton

Renvoie

une référence sur l'instance du singleton

4.24.3.6 getListeTransactions()

```
list< Transaction * > TransactionManager::getListeTransactions ( ) const [noexcept]
```

permet d'obtenir la liste de toute les transactions

Renvoie

la liste de toute les transactions

4.24.3.7 getListeTransactionsParCompte()

```
list< Transaction * > TransactionManager::getListeTransactionsParCompte (
    int const _id_compte )
```

permet d'obtenir une liste des transactions correspondantes à un même compte

Paramètres

<code>_id_compte</code>	: string const, représente le compte dont on souhaite obtenir les transactions
-------------------------	--

Exceptions

<code>SRCH_EXC_T</code>	aucune transaction ne porte sur le compte demandé
-------------------------	---

Renvoie

une liste de pointeur sur les transactions

4.24.3.8 getListeTransactionsParValide()

```
list< Transaction * > TransactionManager::getListeTransactionsParValide (
    int const _id_compte,
    bool const _valide )
```

permet d'obtenir une liste des transactions correspondantes à un même compte

Paramètres

<code>_id_compte</code>	: string const, représente le compte dont on souhaite obtenir les transactions
<code>_valide</code>	: bool const, la validité des transactions à afficher

Exceptions

<code>SRCH_EXC↔ _T</code>	aucune transaction de cette validité ne porte sur le compte demandé
-------------------------------	---

Renvoie

une liste de pointeur sur les transactions

4.24.3.9 getTransaction()

```
Transaction * TransactionManager::getTransaction (
    string const _ref ) [noexcept]
```

permet d'accéder à une transaction depuis sa référence

Paramètres

<code>_ref</code>	: string const, représente la référence de la transaction que l'on souhaite obtenir
-------------------	---

Renvoie

un pointeur sur la transaction si elle existe, nullptr sinon

4.24.3.10 sauver()

```
void TransactionManager::sauver (
    string const _path )
```

permet de sauvegarder un fichier

Paramètres

<code>_ref</code>	: string const, le chemin du fichier de sauvegarde
-------------------	--

Exceptions

<code>UNFD_EXC↔ _F</code>	exception d'accès au fichier
-------------------------------	------------------------------

4.24.3.11 supprimerTransaction()

```
void TransactionManager::supprimerTransaction (
    string const _ref )
```

permet de supprimer une transaction

Paramètres

<code>_ref</code>	: string const, la référence de la transaction à supprimer
-------------------	--

Exceptions

<code>RULE_EXC↔ _T</code>	tentative de suppression sur une référence rapprochée
<code>SRCH_EXC↔ _T</code>	référence _ref inconnue

4.24.4 Documentation des données membres

4.24.4.1 instance

```
TransactionManager TransactionManager::instance [static], [private]
```

l'instance du manager

4.24.4.2 path

```
string TransactionManager::path [private]
```

chemin vers le fichier de sauvegarde utilisé

4.24.4.3 transactions

```
list< Transaction * > TransactionManager::transactions [static], [private]
```

ensemble des transactions existantes

La documentation de cette classe a été générée à partir du fichier suivant :

- [transactionmanager.h](#)
- [transactionmanager.cpp](#)

4.25 Référence de la classe Visiteur

Classe mères de différents visiteurs. Les visiteurs permettent de parcourir la hiérarchie des comptes.

```
#include <Visiteur.h>
```

Dérivée par [VisiteurAffichage](#), [VisiteurFree](#), [VisiteurGetSolde](#), [VisiteurPere](#), [VisiteurPoste](#), et [VisiteurRecherche](#).

Fonctions membres publiques

- virtual void [visiter](#) ([CompteReel](#) &_c) noexcept=0
Méthode qui permet au visiteur de visiter un [CompteReel](#).
- virtual void [visiter](#) ([CompteVirtuel](#) &_c) noexcept=0
Méthode qui permet au visiteur de visiter un [CompteVirtuel](#).

4.25.1 Description détaillée

Classe mères de différents visiteurs. Les visiteurs permettent de parcourir la hiérarchie des comptes.

4.25.2 Documentation des fonctions membres

4.25.2.1 visiter() [1/2]

```
void Visiteur::visiter (
    CompteReel & _c ) [pure virtual], [noexcept]
```

Méthode qui permet au visiteur de visiter un [CompteReel](#).

Paramètres

↩	: CompteReel &, CompteReel auquel le visiteur veut avoir accès.
↩	
C	

Implémenté dans [VisiteurFree](#), [VisiteurGetSolde](#), [VisiteurPere](#), [VisiteurRecherche](#), [VisiteurPoste](#), et [VisiteurAffichage](#).

4.25.2.2 visiter() [2/2]

```
void Visiteur::visiter (
    CompteVirtuel & _c ) [pure virtual], [noexcept]
```

Méthode qui permet au visiteur de visiter un [CompteVirtuel](#).

Paramètres

↔	: CompteVirtuel &, CompteVirtuel auquel le visiteur veut avoir accès.
↔	
↔	
c	

Implémenté dans [VisiteurGetSolde](#), [VisiteurPere](#), [VisiteurFree](#), [VisiteurRecherche](#), [VisiteurPoste](#), et [VisiteurAffichage](#).

La documentation de cette classe a été générée à partir du fichier suivant :

— [Visiteur.h](#)

4.26 Référence de la classe VisiteurAffichage

[VisiteurAffichage](#) est un visiteur qui permet d'afficher la hiérarchie des comptes.

```
#include <VisiteurAffichage.h>
```

Est dérivée de [Visiteur](#).

Fonctions membres publiques

- void [visiter](#) ([CompteReel](#) &_c) noexcept override
Méthode qui permet d'afficher la première lettre du nom du poste du compte suivie du nom d'un compte virtuel grâce à [afficherLigne](#).
- void [visiter](#) ([CompteVirtuel](#) &_c) noexcept override
Méthode qui permet au visiteur de visiter un [CompteVirtuel](#).
- void [afficherLigne](#) (string _s) noexcept
Méthode qui permet d'afficher les informations comptenu dans _s sur une ligne.

Fonctions membres privées

- [VisiteurAffichage](#) ()=default
Constructeur privé
- [VisiteurAffichage](#) (const [VisiteurAffichage](#) &_v)=default
Constructeur par recopie privé.
- [VisiteurAffichage](#) & operator= (const [VisiteurAffichage](#) &_v)=default
Surcharge de l'opérateur d'affectation privé.

Attributs privés

- friend [CompteManager](#)
- int [indentation](#) = 0
permet de gérer l'indentation pour l'affichage des comptes.

4.26.1 Description détaillée

[VisiteurAffichage](#) est un visiteur qui permet d'afficher la hiérarchie des comptes.

4.26.2 Documentation des constructeurs et destructeur

4.26.2.1 VisiteurAffichage() [1/2]

```
VisiteurAffichage::VisiteurAffichage ( ) [private], [default]
```

Constructeur privé

4.26.2.2 VisiteurAffichage() [2/2]

```
VisiteurAffichage::VisiteurAffichage (
    const VisiteurAffichage & _v ) [private], [default]
```

Constructeur par recopie privé.

Paramètres

↔	: const VisiteurAffichage &, référence constante sur le VisiteurAffichage à recopier.
↔	
v	

4.26.3 Documentation des fonctions membres

4.26.3.1 afficherLigne()

```
void VisiteurAffichage::afficherLigne (
    string _s ) [noexcept]
```

Méthode qui permet d'afficher les informations comptenu dans `_s` sur une ligne.

Paramètres

↔	: string, informations à afficher.
↔	
s	

4.26.3.2 operator=()

```
VisiteurAffichage & VisiteurAffichage::operator= (
    const VisiteurAffichage & _v ) [private], [default]
```

Surcharge de l'opérateur d'affectation privé.

Paramètres

↔	: const VisiteurAffichage &, référence constante sur le VisiteurAffichage à affecter.
↔	
v	

4.26.3.3 visiter() [1/2]

```
void VisiteurAffichage::visiter (
    CompteReel & _c ) [override], [virtual], [noexcept]
```

Méthode qui permet d'afficher la première lettre du nom du poste du compte suivie du nom d'un compte virtuel grâce à `afficherLigne`.

Méthode qui permet d'afficher le nom du compte suivie de virtuel grâce à `afficherLigne` puis d'appeler de permettre au visiteur d'accéder à ses fils.

Paramètres

↔	: CompteReel & _c, CompteReel auquel le visiteur veut avoir accès pour afficher ces informations.
↔	
c	
↔	: CompteVirtuel & _c, CompteVirtuel auquel le visiteur veut avoir accès pour afficher ces informations.
↔	
c	

Implémente [Visiteur](#).

4.26.3.4 visiter() [2/2]

```
void VisiteurAffichage::visiter (
    CompteVirtuel & _c ) [override], [virtual], [noexcept]
```

Méthode qui permet au visiteur de visiter un [CompteVirtuel](#).

Paramètres

↔	: CompteVirtuel &, CompteVirtuel auquel le visiteur veut avoir accès.
↔	
↔	
c	

Implémente [Visiteur](#).

4.26.4 Documentation des données membres

4.26.4.1 [CompteManager](#)

```
friend VisiteurAffichage::CompteManager [private]
```

4.26.4.2 indentation

```
int VisiteurAffichage::indentation = 0 [private]
```

permet de gérer l'indentation pour l'affichage des comptes.

La documentation de cette classe a été générée à partir du fichier suivant :

- [VisiteurAffichage.h](#)
- [VisiteurAffichage.cpp](#)

4.27 Référence de la classe [VisiteurFree](#)

[VisiteurFree](#) est un visiteur qui permet de libérer la mémoire dans des comptes.

```
#include <VisiteurFree.h>
```

Est dérivée de [Visiteur](#).

Fonctions membres publiques

- void [visiter](#) ([CompteReel](#) &_c) noexcept override
Méthode qui permet de libérer la mémoire associée à un [CompteReel](#).
- void [visiter](#) ([CompteVirtuel](#) &_c) noexcept override
Méthode qui permet de libérer la mémoire associée à un [CompteVirtuel](#) ainsi que la mémoire utilisée par ses fils.

Fonctions membres privées

- [VisiteurFree](#) ()=default
Constructeur privé
- [VisiteurFree](#) (const [VisiteurFree](#) &_v)=default
Constructeur par recopie privé.
- [VisiteurFree](#) & [operator=](#) (const [VisiteurFree](#) &_v)=default
Surcharge de l'opérateur d'affectation privé.

Attributs privés

- friend [CompteManager](#)

4.27.1 Description détaillée

[VisiteurFree](#) est un visiteur qui permet de libérer la mémoire dans des comptes.

4.27.2 Documentation des constructeurs et destructeur

4.27.2.1 VisiteurFree() [1/2]

```
VisiteurFree::VisiteurFree ( ) [private], [default]
```

Constructeur privé

4.27.2.2 VisiteurFree() [2/2]

```
VisiteurFree::VisiteurFree (
    const VisiteurFree & _v ) [private], [default]
```

Constructeur par recopie privé.

Paramètres

↩	: const VisiteurFree &, référence constante sur le VisiteurFree à recopier.
_↩	
v	

4.27.3 Documentation des fonctions membres

4.27.3.1 operator=()

```
VisiteurFree & VisiteurFree::operator= (
    const VisiteurFree & _v ) [private], [default]
```

Surcharge de l'opérateur d'affectation privé.

Paramètres

↔	: const VisiteurFree &, référence constante sur le VisiteurFree à affecter.
↔	
<u>↔</u>	
<u>v</u>	

4.27.3.2 visiter() [1/2]

```
void VisiteurFree::visiter (
    CompteReel & _c ) [override], [virtual], [noexcept]
```

Méthode qui permet de libérer la mémoire associée à un [CompteReel](#).

Paramètres

↔	: CompteReel & _c, CompteReel auquel le visiteur veut avoir accès pour libérer sa mémoire.
↔	
<u>↔</u>	
<u>c</u>	

Implémente [Visiteur](#).

4.27.3.3 visiter() [2/2]

```
void VisiteurFree::visiter (
    CompteVirtuel & _c ) [override], [virtual], [noexcept]
```

Méthode qui permet de libérer la mémoire associée à un [CompteVirtuel](#) ainsi que la mémoire utilisée par ses fils.

Paramètres

↔	: CompteVirtuel & _c, CompteVirtuel auquel le visiteur veut avoir accès pour libérer sa mémoire et celle de ses fils.
↔	
<u>↔</u>	
<u>c</u>	

Implémente [Visiteur](#).

4.27.4 Documentation des données membres

4.27.4.1 CompteManager

```
friend VisiteurFree::CompteManager [private]
```

La documentation de cette classe a été générée à partir du fichier suivant :

- [VisiteurFree.h](#)
- [VisiteurFree.cpp](#)

4.28 Référence de la classe VisiteurGetSolde

[VisiteurGetSolde](#) est un visiteur qui permet de libérer la mémoire dans des comptes.

```
#include <VisiteurGetSolde.h>
```

Est dérivée de [Visiteur](#).

Fonctions membres publiques

- void [visiter](#) ([CompteReel](#) &_c) noexcept override
Méthode qui permet de calculer le solde d'un compte réel.
- void [visiter](#) ([CompteVirtuel](#) &_c) noexcept override
Méthode qui permet de calculer le solde d'un compte virtuel en faisant la somme des soldes de ses comptes fils.
- double [getSolde](#) () const noexcept
Méthode qui permet d'obtenir le solde du compte demandé s'il existe.

Fonctions membres privées

- [VisiteurGetSolde](#) ()=default
Constructeur privé.
- [VisiteurGetSolde](#) (const [VisiteurGetSolde](#) &_v)=default
Constructeur par recopie privé
- [VisiteurGetSolde](#) & [operator=](#) (const [VisiteurGetSolde](#) &_v)=default
Surcharge de l'opérateur d'affectation privé.

Attributs privés

- friend [CompteManager](#)
- double [solde](#)
Solde du compte.

4.28.1 Description détaillée

[VisiteurGetSolde](#) est un visiteur qui permet de libérer la mémoire dans des comptes.

4.28.2 Documentation des constructeurs et destructeur

4.28.2.1 VisiteurGetSolde() [1/2]

```
VisiteurGetSolde::VisiteurGetSolde ( ) [private], [default]
```

Constructeur privé.

4.28.2.2 VisiteurGetSolde() [2/2]

```
VisiteurGetSolde::VisiteurGetSolde (
    const VisiteurGetSolde & _v ) [private], [default]
```

Constructeur par recopie privé

Paramètres

↔	: const VisiteurGetSolde &, référence constante sur le VisiteurGetSolde à recopier.
↔	
↔	
↔	

4.28.3 Documentation des fonctions membres

4.28.3.1 getSolde()

```
int VisiteurGetSolde::getSolde ( ) const [inline], [noexcept]
```

Méthode qui permet d'obtenir le solde du compte demandé s'il existe.

Renvoie

Le solde du compte si le nom correspond à un compte existant déclenche une exception sinon.

4.28.3.2 operator=()

```
VisiteurGetSolde & VisiteurGetSolde::operator= (
    const VisiteurGetSolde & _v ) [private], [default]
```

Surcharge de l'opérateur d'affectation privé.

Paramètres

↩	: const VisiteurGetSolde &, référence constante sur le VisiteurGetSolde à affecter.
↩	
V	

4.28.3.3 visiter() [1/2]

```
void VisiteurGetSolde::visiter (
    CompteReel & _c ) [override], [virtual], [noexcept]
```

Méthode qui permet de calculer le solde d'un compte réel.

Paramètres

↩	: CompteReel & _c, CompteReel auquel le visiteur veut avoir accès pour calculer son solde.
↩	
C	

Implémente [Visiteur](#).

4.28.3.4 visiter() [2/2]

```
void VisiteurGetSolde::visiter (
    CompteVirtuel & _c ) [override], [virtual], [noexcept]
```

Méthode qui permet de calculer le solde d'un compte virtuel en faisant la somme des soldes de ses comptes fils.

Paramètres

↩	: CompteVirtuel & _c, CompteVirtuel auquel le visiteur veut avoir accès pour calculer son solde.
↩	
C	

Implémente [Visiteur](#).

4.28.4 Documentation des données membres**4.28.4.1 CompteManager**

```
friend VisiteurGetSolde::CompteManager [private]
```

4.28.4.2 solde

```
int VisiteurGetSolde::solde [private]
```

Solde du compte.

La documentation de cette classe a été générée à partir du fichier suivant :

- [VisiteurGetSolde.h](#)
- [VisiteurGetSolde.cpp](#)

4.29 Référence de la classe VisiteurPere

[VisiteurPere](#) est un visiteur qui permet de trouver le père d'un compte.

```
#include <VisiteurPere.h>
```

Est dérivée de [Visiteur](#).

Fonctions membres publiques

- void [visiter](#) ([CompteReel](#) &_c) noexcept override
Méthode pour les comptes réels. Ne fait rien car un compte réel ne peut pas être père.
- void [visiter](#) ([CompteVirtuel](#) &_c) noexcept override
Méthode qui permet de rechercher si le compte cherché est dans les fils.
- [CompteHierarchie](#) * [getResultat](#) () const noexcept
Méthode qui permet d'obtenir un pointeur sur le compte recherché s'il existe.

Fonctions membres privées

- [VisiteurPere](#) ([CompteHierarchie](#) *_recherche) noexcept
- [VisiteurPere](#) (const [VisiteurPere](#) &_v)=default
Constructeur par recopie privé
- [VisiteurPere](#) & [operator=](#) (const [VisiteurPere](#) &_v)=default
Surcharge de l'opérateur d'affectation privé

Attributs privés

- friend [CompteManager](#)
- [CompteHierarchie](#) * [recherche](#)
Pointeur sur le compte recherché.
- [CompteHierarchie](#) * [resultat](#)
Pointeur sur le compte recherché lorsqu'il a été trouvé.

4.29.1 Description détaillée

[VisiteurPere](#) est un visiteur qui permet de trouver le père d'un compte.

4.29.2 Documentation des constructeurs et destructeur

4.29.2.1 VisiteurPere() [1/2]

```
VisiteurPere::VisiteurPere (
    CompteHierarchie * _recherche ) [inline], [private], [noexcept]
```

4.29.2.2 VisiteurPere() [2/2]

```
VisiteurPere::VisiteurPere (
    const VisiteurPere & _v ) [private], [default]
```

Constructeur par recopie privé

Paramètres

↔	: const VisiteurPere &, référence constante sur le VisiteurPere à recopier.
↔	
↔	
↔	

4.29.3 Documentation des fonctions membres**4.29.3.1 getResultat()**

```
CompteHierarchie * VisiteurPere::getResultat ( ) const [inline], [noexcept]
```

Méthode qui permet d'obtenir un pointeur sur le compte recherché s'il existe.

Renvoie

Un pointeur sur le [CompteHierarchie](#) si le père existe, nullptr sinon.

4.29.3.2 operator=()

```
VisiteurPere & VisiteurPere::operator= (
    const VisiteurPere & _v ) [private], [default]
```

Surcharge de l'opérateur d'affectation privé

Paramètres

↔	: const VisiteurPere &, référence constante sur le VisiteurPere à affecter.
↔	
↔	
↔	

4.29.3.3 visiter() [1/2]

```
void VisiteurPere::visiter (
    CompteReel & _c ) [override], [virtual], [noexcept]
```

Méthode pour les comptes réels. Ne fait rien car un compte réel ne peut pas être père.

Paramètres

↔	: CompteReel &
_↔	_c
c	

Implémente [Visiteur](#).

4.29.3.4 visiter() [2/2]

```
void VisiteurPere::visiter (
    CompteVirtuel & _c ) [override], [virtual], [noexcept]
```

Méthode qui permet de rechercher si le compte cherché est dans les fils.

Paramètres

↔	: CompteVirtuel & _c, CompteVirtuel auquel le visiteur veut avoir accès pour comparer ses fils au compte
_↔	recherché.
c	

Implémente [Visiteur](#).

4.29.4 Documentation des données membres

4.29.4.1 [CompteManager](#)

```
friend VisiteurPere::CompteManager [private]
```

4.29.4.2 recherche

```
CompteHierarchie * VisiteurPere::recherche [private]
```

Pointeur sur le compte recherché.

4.29.4.3 resultat

```
CompteHierarchie VisiteurPere::resultat [private]
```

Pointeur sur le compte recherché lorsqu'il a été trouvé.

La documentation de cette classe a été générée à partir du fichier suivant :

- [VisiteurPere.h](#)
- [VisiteurPere.cpp](#)

4.30 Référence de la classe VisiteurPoste

[VisiteurPoste](#) est un visiteur qui permet de récupérer une liste de tous les comptes réels d'un poste.

```
#include <VisiteurPoste.h>
```

Est dérivée de [Visiteur](#).

Fonctions membres publiques

- void [visiter](#) ([CompteReel](#) &_c) noexcept override
Méthode qui permet de vérifier si un [CompteReel](#) est du poste recherché
- void [visiter](#) ([CompteVirtuel](#) &_c) noexcept override
Méthode qui permet de recherche un [Compte](#) parmi les fils du compte passé en paramètre.
- std::list< [CompteHierarchie](#) * > [getResultat](#) () const noexcept
Méthode qui permet d'obtenir la liste des comptes de ce poste.
- void [changePoste](#) ([Poste](#) _p) noexcept
Permet de changer le poste recherché.

Fonctions membres privées

- [VisiteurPoste](#) ([Poste](#) _p) noexcept
- [VisiteurPoste](#) (const [VisiteurPoste](#) &_v)=default
Constructeur par copie privé
- [VisiteurPoste](#) & [operator=](#) (const [VisiteurPoste](#) &_v)=default
Surcharge de l'opérateur d'affectation privé

Attributs privés

- friend [CompteManager](#)
- [Poste](#) p
- std::list< [CompteHierarchie](#) * > [resultat](#)
Liste des comptes correspondants.

4.30.1 Description détaillée

[VisiteurPoste](#) est un visiteur qui permet de récupérer une liste de tous les comptes réels d'un poste.

4.30.2 Documentation des constructeurs et destructeur

4.30.2.1 VisiteurPoste() [1/2]

```
VisiteurPoste::VisiteurPoste (
    Poste _p ) [inline], [private], [noexcept]
```

4.30.2.2 VisiteurPoste() [2/2]

```
VisiteurPoste::VisiteurPoste (
    const VisiteurPoste & _v ) [private], [default]
```

Constructeur par recopie privé

Paramètres

↔	: const VisiteurPoste &, référence constante sur le VisiteurPoste à recopier.
↔	
v	

4.30.3 Documentation des fonctions membres

4.30.3.1 changePoste()

```
void VisiteurPoste::changePoste (
    Poste _p ) [inline], [noexcept]
```

Permet de changer le poste recherché.

4.30.3.2 getResultat()

```
std::list< CompteHierarchie * > VisiteurPoste::getResultat ( ) const [inline], [noexcept]
```

Méthode qui permet d'obtenir la liste des comptes de ce poste.

Renvoie

Une liste contenant les comptes de ce poste.

4.30.3.3 operator=()

```
VisiteurPoste & VisiteurPoste::operator= (
    const VisiteurPoste & _v ) [private], [default]
```

Surcharge de l'opérateur d'affectation privé

Paramètres

↔	: const VisiteurPoste &, référence constante sur le VisiteurPoste à affecter.
_↔	
V	

4.30.3.4 visiter() [1/2]

```
void VisiteurPoste::visiter (
    CompteReel & _c ) [override], [virtual], [noexcept]
```

Méthode qui permet de vérifier si un [CompteReel](#) est du poste recherché

Paramètres

↔	: CompteReel & _c, CompteReel auquel le visiteur veut avoir accès pour le comparer au poste recherché.
_↔	
C	

Implémente [Visiteur](#).

4.30.3.5 visiter() [2/2]

```
void VisiteurPoste::visiter (
    CompteVirtuel & _c ) [override], [virtual], [noexcept]
```

Méthode qui permet de recherche un Compte parmi les fils du compte passé en paramètre.

Paramètres

↔	: CompteVirtuel & _c, CompteVirtuel auquel le visiteur veut avoir accès pour comparer ses fils au poste
_↔	cherché.
C	

Implémente [Visiteur](#).

4.30.4 Documentation des données membres**4.30.4.1 CompteManager**

```
friend VisiteurPoste::CompteManager [private]
```

4.30.4.2 p

```
Poste VisiteurPoste::p [private]
```

4.30.4.3 resultat

```
CompteHierarchie VisiteurPoste::resultat [private]
```

Liste des comptes correspondants.

La documentation de cette classe a été générée à partir du fichier suivant :

- [VisiteurPoste.h](#)
- [VisiteurPoste.cpp](#)

4.31 Référence de la classe VisiteurRecherche

VisiteurRecherche est un visiteur qui permet de recherche un compte dans la hiérarchie des comptes.

```
#include <VisiteurRecherche.h>
```

Est dérivée de [Visiteur](#).

Fonctions membres publiques

- void [visiter](#) ([CompteReel](#) & _c) noexcept override
Méthode qui permet de vérifier si un [CompteReel](#) est le compte recherché.
- void [visiter](#) ([CompteVirtuel](#) & _c) noexcept override
Méthode qui permet de recherche un [Compte](#) parmi les fils du compte passé en paramètre et lui même.
- [CompteHierarchie](#) * [getResultat](#) () const
Méthode qui permet d'obtenir un pointeur sur le compte recherché s'il existe.

Fonctions membres privées

- [VisiteurRecherche](#) (const int _id) noexcept
- [VisiteurRecherche](#) (const [VisiteurRecherche](#) & _v)=default
Constructeur par recopie privé
- [VisiteurRecherche](#) & [operator=](#) (const [VisiteurRecherche](#) & _v)=default
Surcharge de l'opérateur d'affectation privé

Attributs privés

- friend [CompteManager](#)
- int [idcompte](#)
Id du compte recherché.
- [CompteHierarchie](#) * [resultat](#)
Pointeur sur le compte recherché lorsqu'il a été trouvé.

4.31.1 Description détaillée

VisiteurRecherche est un visiteur qui permet de recherche un compte dans la hiérarchie des comptes.

4.31.2 Documentation des constructeurs et destructeur

4.31.2.1 VisiteurRecherche() [1/2]

```
VisiteurRecherche::VisiteurRecherche (
    const int _id ) [inline], [private], [noexcept]
```

4.31.2.2 VisiteurRecherche() [2/2]

```
VisiteurRecherche::VisiteurRecherche (
    const VisiteurRecherche & _v ) [private], [default]
```

Constructeur par copie privé

Paramètres

↔	: const VisiteurRecherche &, référence constante sur le VisiteurRecherche à copier.
↔	
v	

4.31.3 Documentation des fonctions membres

4.31.3.1 getResultat()

```
CompteHierarchie * VisiteurRecherche::getResultat ( ) const [inline]
```

Méthode qui permet d'obtenir un pointeur sur le compte recherché s'il existe.

Renvoie

Un pointeur sur le [CompteHierarchie](#) si le nom correspond à un compte existant, nullptr sinon.

4.31.3.2 operator=()

```
VisiteurRecherche & VisiteurRecherche::operator= (
    const VisiteurRecherche & _v ) [private], [default]
```

Surcharge de l'opérateur d'affectation privé

Paramètres

↔	: const VisiteurRecherche &, référence constante sur le VisiteurRecherche à affecter.
_↔	
v	

4.31.3.3 visiter() [1/2]

```
void VisiteurRecherche::visiter (
    CompteReel & _c ) [override], [virtual], [noexcept]
```

Méthode qui permet de vérifier si un [CompteReel](#) est le compte recherché.

Paramètres

↔	: CompteReel & _c, CompteReel auquel le visiteur veut avoir accès pour le comparer au compte recherché.
_↔	
c	

Implémente [Visiteur](#).

4.31.3.4 visiter() [2/2]

```
void VisiteurRecherche::visiter (
    CompteVirtuel & _c ) [override], [virtual], [noexcept]
```

Méthode qui permet de recherche un [Compte](#) parmi les fils du compte passé en paramètre et lui même.

Paramètres

↔	: CompteVirtuel & _c, CompteVirtuel auquel le visiteur veut avoir accès pour comparer ses fils et lui même au compte recherché.
_↔	
c	

Implémente [Visiteur](#).

4.31.4 Documentation des données membres**4.31.4.1 CompteManager**

```
friend VisiteurRecherche::CompteManager [private]
```


4.31.4.2 idcompte

```
int VisiteurRecherche::idcompte [private]
```

Id du compte recherché.

4.31.4.3 resultat

```
CompteHierarchie VisiteurRecherche::resultat [private]
```

Pointeur sur le compte recherché lorsqu'il a été trouvé.

La documentation de cette classe a été générée à partir du fichier suivant :

- [VisiteurRecherche.h](#)
- [VisiteurRecherche.cpp](#)

Chapitre 5

Documentation des fichiers

5.1 Référence du fichier Actif.cpp

```
#include "../header/CompteManager.h"
```

5.2 Référence du fichier Actif.h

```
#include "../header/Passif.h"
```

Classes

- class [Actif](#)
[Actif](#) permet de représenter les propriétés des comptes de poste [Actif](#).

5.3 Référence du fichier CompteHierarchie.cpp

```
#include "../header/CompteManager.h"
```

5.4 Référence du fichier CompteHierarchie.h

```
#include "../header/Recette.h"
```

Classes

- class [CompteHierarchie](#)
[CompteHierarchie](#) est la classe mère pour les éléments de la hiérarchie des comptes qui permet de manipuler tous les comptes via la même interface.

Énumérations

- enum `Type` { `VIRTUEL` = 0, `REEL` = 1 }
représente les deux types de Compte : *Virtuel* (c'est-à-dire un compte qui n'est destiné qu'à regrouper des sous-comptes sans enregistrer aucune opération) et *Réel* (les comptes sur lesquels seront enregistrées les opérations).

5.4.1 Documentation du type de l'énumération

5.4.1.1 Type

enum `Type`

représente les deux types de Compte : *Virtuel* (c'est-à-dire un compte qui n'est destiné qu'à regrouper des sous-comptes sans enregistrer aucune opération) et *Réel* (les comptes sur lesquels seront enregistrées les opérations).

Valeurs énumérées

VIRTUEL	
REEL	

5.5 Référence du fichier `CompteManager.cpp`

```
#include "../header/CompteManager.h"
#include <ctime>
```

5.6 Référence du fichier `CompteManager.h`

```
#include "../../Transaction/header/transactionmanager.h"
```

Classes

- class `CompteManager`
CompteManager gère l'ensemble des Comptes.

5.7 Référence du fichier `CompteReel.cpp`

```
#include "../header/CompteManager.h"
```

5.8 Référence du fichier CompteReel.h

```
#include "../header/CompteVirtuel.h"  
#include <iostream>  
#include <list>  
#include <ctime>
```

Classes

- class [CompteReel](#)
CompteReel représente les compte réels (les comptes sur lesquels auront lieu les opérations).

5.9 Référence du fichier CompteVirtuel.cpp

```
#include "../header/CompteManager.h"
```

5.10 Référence du fichier CompteVirtuel.h

```
#include "../header/VisiteurRecherche.h"
```

Classes

- class [CompteVirtuel](#)
CompteVirtuel représente les Compte Virtuels (comptes destinés à regrouper des sous-compte sans enregistrer d'opération).

5.11 Référence du fichier declarations.h

```
#include <iostream>  
#include <list>  
#include <memory>  
#include <time.h>  
#include <ctime>  
#include <conio.h>  
#include <set>  
#include <iomanip>
```

5.12 Référence du fichier Depense.cpp

```
#include "../header/CompteManager.h"
```

5.13 Référence du fichier Depense.h

```
#include "../header/Actif.h"
```

Classes

- class [Depense](#)
[Depense](#) permet de représenter les propriétés des comptes de poste [Depense](#).

5.14 Référence du fichier Exception.h

```
#include "../../Transaction/header/declarations.h"
```

Classes

- class [Exception](#)
[Exception](#) permet de gérer une erreur ainsi que son message.

5.15 Référence du fichier ExceptionCompte.h

```
#include "Exception.h"
```

Classes

- class [ExceptionHierarchie](#)
[ExceptionHierarchie](#) est créée lors d'une opération illégale sur la hiérarchie de comptes.
- class [ExceptionComptabilite](#)
[ExceptionComptabilite](#) est créée lors d'une opération de comptabilité illégale.

5.16 Référence du fichier exceptiontransaction.h

```
#include "../../Compte/header/ExceptionCompte.h"
```

Classes

- class [ExceptionTransaction](#)
traite les exceptions de la partie transaction
- class [ExceptionFichier](#)
traite les exceptions de la partie fichier

5.17 Référence du fichier HierarchieBuilder.cpp

```
#include "../header/CompteManager.h"
```

5.18 Référence du fichier HierarchieBuilder.h

```
#include "../../../Transaction/header/transactionbuilder.h"
```

Classes

- class [HierarchieBuilder](#)
[HierarchieBuilder](#) gère la création de la hiérarchie des comptes.

5.19 Référence du fichier main.cpp

```
#include "../Compte/header/CompteManager.h"
```

Macros

- #define [BUFF_DATE](#) 10
- #define [BUFF_REF](#) 16
- #define [BUFF_INTITUTLE](#) 20
- #define [BUFF_COMPTE](#) 20
- #define [BUFF_CREDIT](#) 10
- #define [BUFF_DEBIT](#) 10

Fonctions

- string [buff](#) (string text, size_t buffer)
- void [afficherLigneTransaction](#) ([Transaction](#) *const _transaction, bool _est_selectionnee, [CompteManager](#) &_c_comp_manager)
- int [main](#) ()

5.19.1 Documentation des macros

5.19.1.1 BUFF_COMPTE

```
#define BUFF_COMPTE 20
```

5.19.1.2 BUFF_CREDIT

```
#define BUFF_CREDIT 10
```

5.19.1.3 BUFF_DATE

```
#define BUFF_DATE 10
```

5.19.1.4 BUFF_DEBIT

```
#define BUFF_DEBIT 10
```

5.19.1.5 BUFF_INTITUTLE

```
#define BUFF_INTITUTLE 20
```

5.19.1.6 BUFF_REF

```
#define BUFF_REF 16
```

5.19.2 Documentation des fonctions

5.19.2.1 afficherLigneTransaction()

```
void afficherLigneTransaction (
    Transaction *const _transaction,
    bool _est_selectionnee,
    CompteurManager & _c_comp_manager )
```

5.19.2.2 buff()

```
string buff (
    string text,
    size_t buffer )
```


5.19.2.3 main()

```
int main ( )
```

5.20 Référence du fichier Memento.cpp

```
#include "../header/CompteManager.h"
```

5.21 Référence du fichier Memento.h

```
#include "../../../Transaction/header/exceptiontransaction.h"
```

Classes

— class [Memento](#)

Le [Memento](#) permet de sauvegarder la date et le solde du dernier rapprochement.

5.22 Référence du fichier operation.cpp

```
#include "../../../Compte/header/CompteManager.h"
```

Fonctions

— `list< Operation * > operator<< (list< Operation * > &_listOperations, Operation *_operation) noexcept`

5.22.1 Documentation des fonctions

5.22.1.1 operator<<()

```
list<Operation*> operator<< (  
    list< Operation * > &_listOperations,  
    Operation *_operation ) [noexcept]
```

5.23 Référence du fichier operation.h

```
#include "../../../Compte/header/CompteReel.h"
```

Classes

- class [Operation](#)
Objet représentant une opération comptable.

5.24 Référence du fichier Originator.h

```
#include "../header/Memento.h"
```

Classes

- class [Originator](#)
[Originator](#) permet de manipuler les mementos.

5.25 Référence du fichier Passif.cpp

```
#include "../header/CompteManager.h"
```

5.26 Référence du fichier Passif.h

```
#include "../header/PostePR.h"
```

Classes

- class [Passif](#)
[Passif](#) permet de représenter les propriétés des comptes de poste [Passif](#).

5.27 Référence du fichier PosteAD.h

```
#include "../header/Racine.h"
```

Classes

- class [PosteAD](#)
[PosteAD](#) est la classe mère des postes de compte : [Actif](#) et [Depense](#). Elle a pour but de définir les méthodes créditer et débiter qui sont communes à ces deux classes.

5.28 Référence du fichier PosteCompte.cpp

```
#include "../header/CompteManager.h"
```

5.29 Référence du fichier PosteCompte.h

```
#include "../header/Originator.h"
```

Classes

- class [PosteCompte](#)
[PosteCompte](#) est la classe mère des différents postes de compte : [Actif](#) et [Dépense](#), [Passif](#) et [Recette](#), et la [Racine](#).

Énumérations

- enum [Poste](#) {
 [PASSIF](#), [ACTIF](#), [DEPENSE](#), [RECETTE](#),
 [RACINE](#) }

5.29.1 Documentation du type de l'énumération

5.29.1.1 Poste

```
enum Poste
```

{PASSIF, ACTIF, DEPENSE, RECETTE, RACINE}, représente le nom du poste d'un Compte : [Passif](#), [Actif](#), [Depense](#), [Recette](#) et la [Racine](#).

Valeurs énumérées

PASSIF	
ACTIF	
DEPENSE	
RECETTE	
RACINE	

5.30 Référence du fichier PostePR.h

```
#include "../header/PosteAD.h"
```

Classes

- class [PostePR](#)

[PostePR](#) est la classe mère des postes de compte : [Recette](#) et [Passif](#). Elle a pour but de définir les méthodes créditer et débiter qui sont communes à ces deux classes.

5.31 Référence du fichier Racine.cpp

```
#include "../header/CompteManager.h"
```

5.32 Référence du fichier Racine.h

```
#include "../header/PosteCompte.h"
```

Classes

- class [Racine](#)

[Racine](#) permet de représenter le comportement d'un compte [Racine](#), c'est-à-dire celui qui contient tous les autres comptes.

5.33 Référence du fichier Recette.cpp

```
#include "../header/CompteManager.h"
```

5.34 Référence du fichier Recette.h

```
#include "../header/Depense.h"
```

Classes

- class [Recette](#)

[Recette](#) permet de représenter le comportement des comptes de poste [Recette](#).

5.35 Référence du fichier transaction.cpp

```
#include "../../Compte/header/CompteManager.h"
```

5.36 Référence du fichier transaction.h

```
#include "../header/operation.h"
```

Classes

— class [Transaction](#)

La transaction représente un ensemble d'opérations équilibrées effectuées à une même date entre plusieurs comptes.

5.37 Référence du fichier transactionbuilder.cpp

```
#include "../../Compte/header/CompteManager.h"
```

5.38 Référence du fichier transactionbuilder.h

```
#include "../header/transaction.h"
```

Classes

— class [TransactionBuilder](#)

Le transaction builder permet de construire les transactions. Il utilise le pattern builder et singleton.

5.39 Référence du fichier transactionmanager.cpp

```
#include "../../Compte/header/CompteManager.h"
```

5.40 Référence du fichier transactionmanager.h

```
#include "../../Compte/header/HierarchieBuilder.h"
```

Classes

— class [TransactionManager](#)

Le transaction manager permet de gérer les transactions. Elle est basée sur les pattern Manager et Singleton.

5.41 Référence du fichier Visiteur.cpp

```
#include "../header/CompteManager.h"
```

5.42 Référence du fichier Visiteur.h

```
#include "../header/CompteHierarchie.h"
```

Classes

— class [Visiteur](#)

Classe mères de différents visiteurs. Les visiteurs permettent de parcourir la hiérarchie des comptes.

5.43 Référence du fichier VisiteurAffichage.cpp

```
#include "../header/CompteManager.h"
```

5.44 Référence du fichier VisiteurAffichage.h

```
#include "../header/VisiteurGetSolde.h"
```

Classes

— class [VisiteurAffichage](#)

[VisiteurAffichage](#) est un visiteur qui permet d'afficher la hiérarchie des comptes.

5.45 Référence du fichier VisiteurFree.cpp

```
#include "../header/CompteManager.h"
```

5.46 Référence du fichier VisiteurFree.h

```
#include "../header/VisiteurPoste.h"
```

Classes

- class [VisiteurFree](#)
VisiteurFree est un visiteur qui permet de libérer la mémoire dans des comptes.

5.47 Référence du fichier VisiteurGetSolde.cpp

```
#include "../header/CompteManager.h"
```

5.48 Référence du fichier VisiteurGetSolde.h

```
#include "../header/VisiteurFree.h"
```

Classes

- class [VisiteurGetSolde](#)
VisiteurGetSolde est un visiteur qui permet de libérer la mémoire dans des comptes.

5.49 Référence du fichier VisiteurPere.cpp

```
#include "../header/CompteManager.h"
```

5.50 Référence du fichier VisiteurPere.h

```
#include "../header/VisiteurAffichage.h"
```

Classes

- class [VisiteurPere](#)
VisiteurPere est un visiteur qui permet de trouver le père d'un compte.

5.51 Référence du fichier VisiteurPoste.cpp

```
#include "../header/CompteManager.h"  
#include "../header/VisiteurPoste.h"
```

5.52 Référence du fichier VisiteurPoste.h

```
#include "../header/Visiteur.h"
```

Classes

- class [VisiteurPoste](#)
VisiteurPoste est un visiteur qui permet de récupérer une liste de tous les comptes réels d'un poste.

5.53 Référence du fichier VisiteurRecherche.cpp

```
#include "../header/CompteManager.h"
```

5.54 Référence du fichier VisiteurRecherche.h

```
#include "../header/VisiteurPere.h"
```

Classes

- class [VisiteurRecherche](#)
VisiteurRecherche est un visiteur qui permet de recherche un compte dans la hiérarchie des comptes.

Index

- ~Actif
 - Actif, [8](#)
- ~CompteHierarchie
 - CompteHierarchie, [11](#)
- ~CompteManager
 - CompteManager, [21](#)
- ~CompteReel
 - CompteReel, [30](#)
- ~CompteVirtuel
 - CompteVirtuel, [38](#)
- ~Depense
 - Depense, [44](#)
- ~Operation
 - Operation, [58](#)
- ~Passif
 - Passif, [65](#)
- ~Racine
 - Racine, [72](#)
- ~Recette
 - Recette, [75](#)
- ~Transaction
 - Transaction, [79](#)
- ~TransactionBuilder
 - TransactionBuilder, [84](#)
- ~TransactionManager
 - TransactionManager, [88](#)
- accepte
 - CompteHierarchie, [13](#)
 - CompteReel, [31](#)
 - CompteVirtuel, [38](#)
- ACTIF
 - PosteCompte.h, [123](#)
- Actif, [7](#)
 - ~Actif, [8](#)
 - Actif, [8](#)
 - getInstance, [8](#)
 - getLetter, [9](#)
 - getPoste, [9](#)
 - instance, [10](#)
 - operator=, [9](#)
- Actif.cpp, [115](#)
- Actif.h, [115](#)
- affiche
 - CompteManager, [21](#)
- afficherLigne
 - VisiteurAffichage, [96](#)
- afficherLigneTransaction
 - main.cpp, [120](#)
- ajouterFils
 - CompteHierarchie, [13](#)
 - CompteManager, [22](#)
 - CompteReel, [31](#)
 - CompteVirtuel, [39](#)
- ajouterOperation
 - TransactionManager, [88](#)
- ajouterTransaction
 - TransactionManager, [89](#)
- bilan
 - CompteManager, [22](#)
- buff
 - main.cpp, [120](#)
- BUFF_COMPTE
 - main.cpp, [119](#)
- BUFF_CREDIT
 - main.cpp, [119](#)
- BUFF_DATE
 - main.cpp, [120](#)
- BUFF_DEBIT
 - main.cpp, [120](#)
- BUFF_INITITITLE
 - main.cpp, [120](#)
- BUFF_REF
 - main.cpp, [120](#)
- changePoste
 - VisiteurPoste, [108](#)
- charger
 - CompteManager, [22](#)
 - TransactionManager, [89](#)
- cloturer
 - CompteManager, [23](#)
- code_erreur
 - Exception, [47](#)
- CodeExcepComptabilite
 - ExceptionComptabilite, [48](#)
- CodeExcepFichier
 - ExceptionFichier, [49](#)
- CodeExcepHierarchie
 - ExceptionHierarchie, [51](#)
- CodeExcepTransaction
 - ExceptionTransaction, [52](#)
- compte
 - Operation, [61](#)
- CompteHierarchie, [10](#)
 - ~CompteHierarchie, [11](#)
 - accepte, [13](#)
 - ajouterFils, [13](#)
 - CompteHierarchie, [11, 12](#)

- CompteManager, 18
- fils, 18
- getDernierMemento, 13
- getFils, 14
- getId, 14
- getMementos, 14
- getNom, 15
- getPoste, 15
- getPremierMemento, 15
- getType, 15
- HierarchieBuilder, 18
- id, 19
- nom, 19
- operator=, 16
- poste, 19
- saveState, 16
- supprimerCompte, 16
- supprimerFils, 17
- toString, 17
- undo, 17
- VisiteurFree, 18
- CompteHierarchie.cpp, 115
- CompteHierarchie.h, 115
 - REEL, 116
 - Type, 116
 - VIRTUEL, 116
- CompteManager, 19
 - ~CompteManager, 21
 - affiche, 21
 - ajouterFils, 22
 - bilan, 22
 - charger, 22
 - cloturer, 23
 - CompteHierarchie, 18
 - CompteManager, 21
 - CompteReel, 35
 - CompteVirtuel, 42
 - crediterCompte, 23
 - debiterCompte, 23
 - deplacerFils, 24
 - getCompte, 24
 - getInstance, 25
 - getOriginator, 25
 - getPere, 25
 - getSolde, 26
 - HierarchieBuilder, 55
 - instance, 28
 - o, 28
 - operator=, 26
 - path, 29
 - racine, 29
 - rapprocherCompte, 26
 - releve, 27
 - resultat, 27
 - sauver, 27
 - setPath, 28
 - supprimerCompte, 28
 - Transaction, 82
 - VisiteurAffichage, 98
 - VisiteurFree, 100
 - VisiteurGetSolde, 103
 - VisiteurPere, 106
 - VisiteurPoste, 109
 - VisiteurRecherche, 112
- CompteManager.cpp, 116
- CompteManager.h, 116
- CompteReel, 29
 - ~CompteReel, 30
 - accepte, 31
 - ajouterFils, 31
 - CompteManager, 35
 - CompteReel, 30
 - getDernierMemento, 31
 - getMementos, 33
 - getPremierMemento, 33
 - getType, 33
 - HierarchieBuilder, 36
 - mementos, 36
 - operator=, 33
 - saveState, 34
 - supprimerCompte, 34
 - supprimerFils, 34
 - toString, 35
 - undo, 35
 - VisiteurFree, 36
- CompteReel.cpp, 116
- CompteReel.h, 117
- CompteVirtuel, 36
 - ~CompteVirtuel, 38
 - accepte, 38
 - ajouterFils, 39
 - CompteManager, 42
 - CompteVirtuel, 37, 38
 - getDernierMemento, 39
 - getMementos, 39
 - getPremierMemento, 39
 - getType, 40
 - HierarchieBuilder, 42
 - operator=, 40
 - saveState, 40
 - supprimerCompte, 41
 - supprimerFils, 41
 - toString, 42
 - undo, 42
 - VisiteurFree, 42
- CompteVirtuel.cpp, 117
- CompteVirtuel.h, 117
- credit
 - Operation, 61
- crediter
 - PosteAD, 67
 - PosteCompte, 69
 - PostePR, 70
 - Racine, 72
- crediterCompte
 - CompteManager, 23

- creerAvecFichier
 - HierarchieBuilder, [53](#)
- creerEmpty
 - HierarchieBuilder, [54](#)
- creerFils
 - HierarchieBuilder, [54](#)
- creerOperation
 - TransactionBuilder, [84](#)
- creerTransaction
 - TransactionBuilder, [85](#)
- creerTransactionAvecFichier
 - TransactionBuilder, [85](#)
- date
 - Memento, [56](#)
 - Originator, [64](#)
 - Transaction, [82](#)
- debit
 - Operation, [61](#)
- debiter
 - PosteAD, [68](#)
 - PosteCompte, [69](#)
 - PostePR, [70](#)
 - Racine, [73](#)
- debiterCompte
 - CompteManager, [23](#)
- declarations.h, [117](#)
- DEPENSE
 - PosteCompte.h, [123](#)
- Depense, [43](#)
 - ~Depense, [44](#)
 - Depense, [44](#)
 - getInstance, [44](#)
 - getLetter, [44](#)
 - getPoste, [45](#)
 - instance, [45](#)
 - operator=, [45](#)
- Depense.cpp, [117](#)
- Depense.h, [118](#)
- deplacerFils
 - CompteManager, [24](#)
- detruireTransaction
 - TransactionBuilder, [86](#)
- editerTransaction
 - TransactionManager, [90](#)
- Exception, [46](#)
 - code_erreur, [47](#)
 - Exception, [46](#)
 - getCode, [47](#)
 - getMessage, [47](#)
 - message, [47](#)
- Exception.h, [118](#)
- ExceptionComptabilite, [48](#)
 - CodeExcepComptabilite, [48](#)
 - ExceptionComptabilite, [49](#)
 - MEMV_EXC_C, [48](#)
 - ROOT_EXC_C, [48](#)
 - SUPREL_EXC_C, [48](#)
 - SUPVIR_EXC_C, [48](#)
 - UKWN_EXC_C, [48](#)
- ExceptionCompte.h, [118](#)
- ExceptionFichier, [49](#)
 - CodeExcepFichier, [49](#)
 - ExceptionFichier, [50](#)
 - SNTX_EXC_F, [50](#)
 - UFND_EXC_F, [50](#)
 - UKWN_EXC_F, [50](#)
- ExceptionHierarchie, [50](#)
 - CodeExcepHierarchie, [51](#)
 - ExceptionHierarchie, [51](#)
 - FLSR_EXC_H, [51](#)
 - IDNF_EXC_H, [51](#)
 - POSTE_EXC_H, [51](#)
 - RULE_EXC_H, [51](#)
 - TYPE_EXC_H, [51](#)
 - UKWN_EXC_H, [51](#)
- ExceptionTransaction, [51](#)
 - CodeExcepTransaction, [52](#)
 - ExceptionTransaction, [52](#)
 - MEMO_EXC_T, [52](#)
 - NULL_EXC_T, [52](#)
 - RULE_EXC_T, [52](#)
 - SRCH_EXC_T, [52](#)
 - UKWN_EXC_T, [52](#)
- exceptiontransaction.h, [118](#)
- files
 - CompteHierarchie, [18](#)
- FLSR_EXC_H
 - ExceptionHierarchie, [51](#)
- getCode
 - Exception, [47](#)
- getCompte
 - CompteManager, [24](#)
- getCredit
 - Operation, [59](#)
- getDate
 - Memento, [56](#)
 - Originator, [62](#)
 - Transaction, [79](#)
- getDebit
 - Operation, [59](#)
- getDernierMemento
 - CompteHierarchie, [13](#)
 - CompteReel, [31](#)
 - CompteVirtuel, [39](#)
- getFils
 - CompteHierarchie, [14](#)
- getId
 - CompteHierarchie, [14](#)
- getIdCompte
 - Operation, [59](#)
- getInstance
 - Actif, [8](#)
 - CompteManager, [25](#)
 - Depense, [44](#)

- Passif, 66
- Racine, 73
- Recette, 76
- TransactionManager, 90
- getLetter
 - Actif, 9
 - Depense, 44
 - Passif, 66
 - PosteCompte, 69
 - Racine, 73
 - Recette, 76
- getListeOperations
 - Transaction, 79
- getListeTransactions
 - TransactionManager, 91
- getListeTransactionsParCompte
 - TransactionManager, 91
- getListeTransactionsParValide
 - TransactionManager, 91
- getMementos
 - CompteHierarchie, 14
 - CompteReel, 33
 - CompteVirtuel, 39
- getMemo
 - Transaction, 79
- getMessage
 - Exception, 47
- getNom
 - CompteHierarchie, 15
- getOriginator
 - CompteManager, 25
- getPere
 - CompteManager, 25
- getPoste
 - Actif, 9
 - CompteHierarchie, 15
 - Depense, 45
 - Passif, 66
 - PosteCompte, 69
 - Racine, 73
 - Recette, 76
- getPremierMemento
 - CompteHierarchie, 15
 - CompteReel, 33
 - CompteVirtuel, 39
- getReference
 - Transaction, 80
- getResultat
 - VisiteurPere, 105
 - VisiteurPoste, 108
 - VisiteurRecherche, 111
- getSolde
 - CompteManager, 26
 - Memento, 56
 - Originator, 63
 - VisiteurGetSolde, 102
- getStringDate
 - Transaction, 80
- getTransaction
 - TransactionManager, 92
- getType
 - CompteHierarchie, 15
 - CompteReel, 33
 - CompteVirtuel, 40
- getValide
 - Transaction, 80
- HierarchieBuilder, 53
 - CompteHierarchie, 18
 - CompteManager, 55
 - CompteReel, 36
 - CompteVirtuel, 42
 - creerAvecFichier, 53
 - creerEmpty, 54
 - creerFils, 54
 - idcompte, 55
- HierarchieBuilder.cpp, 119
- HierarchieBuilder.h, 119
- id
 - CompteHierarchie, 19
- idcompte
 - HierarchieBuilder, 55
 - VisiteurRecherche, 112
- IDNF_EXC_H
 - ExceptionHierarchie, 51
- indentation
 - VisiteurAffichage, 98
- instance
 - Actif, 10
 - CompteManager, 28
 - Depense, 45
 - Passif, 67
 - Racine, 74
 - Recette, 77
 - TransactionManager, 93
- instance_builder
 - TransactionBuilder, 86
- isCompte
 - Operation, 59
- main
 - main.cpp, 120
- main.cpp, 119
 - afficherLigneTransaction, 120
 - buff, 120
 - BUFF_COMPTE, 119
 - BUFF_CREDIT, 119
 - BUFF_DATE, 120
 - BUFF_DEBIT, 120
 - BUFF_INTITUTLE, 120
 - BUFF_REF, 120
 - main, 120
- Memento, 55
 - date, 56
 - getDate, 56
 - getSolde, 56

- Memento, 56
- solde, 57
- Memento.cpp, 121
- Memento.h, 121
- mementos
 - CompteReel, 36
- memo
 - Transaction, 82
- MEMO_EXC_T
 - ExceptionTransaction, 52
- MEMV_EXC_C
 - ExceptionComptabilite, 48
- message
 - Exception, 47
- nom
 - CompteHierarchie, 19
- NULL_EXC_T
 - ExceptionTransaction, 52
- o
 - CompteManager, 28
- Operation, 57
 - ~Operation, 58
 - compte, 61
 - credit, 61
 - debit, 61
 - getCredit, 59
 - getDebit, 59
 - getIdCompte, 59
 - isCompte, 59
 - Operation, 58
 - operator<, 60
 - operator<<, 60
 - operator<=, 60
 - operator>, 60
 - operator>=, 60
 - transaction, 61
 - TransactionBuilder, 61
- operation.cpp, 121
 - operator<<, 121
- operation.h, 121
- operations
 - Transaction, 82
- operator<
 - Operation, 60
 - Transaction, 80
- operator<<
 - Operation, 60
 - operation.cpp, 121
- operator<=
 - Operation, 60
 - Transaction, 81
- operator>
 - Operation, 60
 - Transaction, 81
- operator>=
 - Operation, 60
 - Transaction, 81
- operator=
 - Actif, 9
 - CompteHierarchie, 16
 - CompteManager, 26
 - CompteReel, 33
 - CompteVirtuel, 40
 - Depense, 45
 - Passif, 66
 - Racine, 74
 - Recette, 76
 - VisiteurAffichage, 97
 - VisiteurFree, 99
 - VisiteurGetSolde, 102
 - VisiteurPere, 105
 - VisiteurPoste, 108
 - VisiteurRecherche, 111
- Originator, 62
 - date, 64
 - getDate, 62
 - getSolde, 63
 - Originator, 62
 - restoreMemento, 63
 - saveState, 63
 - setSolde, 63
 - solde, 64
- Originator.h, 122
- p
 - VisiteurPoste, 109
- PASSIF
 - PosteCompte.h, 123
- Passif, 64
 - ~Passif, 65
 - getInstance, 66
 - getLetter, 66
 - getPoste, 66
 - instance, 67
 - operator=, 66
 - Passif, 65
- Passif.cpp, 122
- Passif.h, 122
- path
 - CompteManager, 29
 - TransactionManager, 93
- Poste
 - PosteCompte.h, 123
- poste
 - CompteHierarchie, 19
- POSTE_EXC_H
 - ExceptionHierarchie, 51
- PosteAD, 67
 - crediter, 67
 - debiter, 68
- PosteAD.h, 122
- PosteCompte, 68
 - crediter, 69
 - debiter, 69
 - getLetter, 69
 - getPoste, 69

- PosteCompte.cpp, [123](#)
- PosteCompte.h, [123](#)
 - ACTIF, [123](#)
 - DEPENSE, [123](#)
 - PASSIF, [123](#)
 - Poste, [123](#)
 - RACINE, [123](#)
 - RECETTE, [123](#)
- PostePR, [70](#)
 - crediter, [70](#)
 - debiter, [70](#)
- PostePR.h, [123](#)
- RACINE
 - PosteCompte.h, [123](#)
- Racine, [71](#)
 - ~Racine, [72](#)
 - crediter, [72](#)
 - debiter, [73](#)
 - getInstance, [73](#)
 - getLetter, [73](#)
 - getPoste, [73](#)
 - instance, [74](#)
 - operator=, [74](#)
 - Racine, [72](#)
- racine
 - CompteManager, [29](#)
- Racine.cpp, [124](#)
- Racine.h, [124](#)
- rapprocherCompte
 - CompteManager, [26](#)
- RECETTE
 - PosteCompte.h, [123](#)
- Recette, [74](#)
 - ~Recette, [75](#)
 - getInstance, [76](#)
 - getLetter, [76](#)
 - getPoste, [76](#)
 - instance, [77](#)
 - operator=, [76](#)
 - Recette, [75](#)
- Recette.cpp, [124](#)
- Recette.h, [124](#)
- recherche
 - VisiteurPere, [106](#)
- REEL
 - CompteHierarchie.h, [116](#)
- reference_transaction
 - Transaction, [82](#)
- releve
 - CompteManager, [27](#)
- restoreMemento
 - Originator, [63](#)
- resultat
 - CompteManager, [27](#)
 - VisiteurPere, [106](#)
 - VisiteurPoste, [110](#)
 - VisiteurRecherche, [113](#)
- ROOT_EXC_C
 - ExceptionComptabilite, [48](#)
- RULE_EXC_H
 - ExceptionHierarchie, [51](#)
- RULE_EXC_T
 - ExceptionTransaction, [52](#)
- sauver
 - CompteManager, [27](#)
 - TransactionManager, [92](#)
- saveState
 - CompteHierarchie, [16](#)
 - CompteReel, [34](#)
 - CompteVirtuel, [40](#)
 - Originator, [63](#)
- setPath
 - CompteManager, [28](#)
- setSolde
 - Originator, [63](#)
- setValide
 - Transaction, [81](#)
- SNTX_EXC_F
 - ExceptionFichier, [50](#)
- solde
 - Memento, [57](#)
 - Originator, [64](#)
 - VisiteurGetSolde, [103](#)
- SRCH_EXC_T
 - ExceptionTransaction, [52](#)
- supprimerCompte
 - CompteHierarchie, [16](#)
 - CompteManager, [28](#)
 - CompteReel, [34](#)
 - CompteVirtuel, [41](#)
- supprimerFils
 - CompteHierarchie, [17](#)
 - CompteReel, [34](#)
 - CompteVirtuel, [41](#)
- supprimerTransaction
 - TransactionManager, [93](#)
- SUPREL_EXC_C
 - ExceptionComptabilite, [48](#)
- SUPVIR_EXC_C
 - ExceptionComptabilite, [48](#)
- toString
 - CompteHierarchie, [17](#)
 - CompteReel, [35](#)
 - CompteVirtuel, [42](#)
- Transaction, [77](#)
 - ~Transaction, [79](#)
 - CompteManager, [82](#)
 - date, [82](#)
 - getDate, [79](#)
 - getListeOperations, [79](#)
 - getMemo, [79](#)
 - getReference, [80](#)
 - getStringDate, [80](#)
 - getValide, [80](#)
 - memo, [82](#)

- operations, 82
- operator<, 80
- operator<=, 81
- operator>, 81
- operator>=, 81
- reference_transaction, 82
- setValide, 81
- Transaction, 78
- TransactionBuilder, 82
- valide, 82
- transaction
 - Operation, 61
- transaction.cpp, 124
- transaction.h, 125
- TransactionBuilder, 83
 - ~TransactionBuilder, 84
 - creerOperation, 84
 - creerTransaction, 85
 - creerTransactionAvecFichier, 85
 - destruireTransaction, 86
 - instance_builder, 86
 - Operation, 61
 - Transaction, 82
 - TransactionBuilder, 84
 - TransactionManager, 86
- transactionbuilder.cpp, 125
- transactionbuilder.h, 125
- TransactionManager, 87
 - ~TransactionManager, 88
 - ajouterOperation, 88
 - ajouterTransaction, 89
 - charger, 89
 - editerTransaction, 90
 - getInstance, 90
 - getListeTransactions, 91
 - getListeTransactionsParCompte, 91
 - getListeTransactionsParValide, 91
 - getTransaction, 92
 - instance, 93
 - path, 93
 - sauver, 92
 - supprimerTransaction, 93
 - TransactionBuilder, 86
 - TransactionManager, 88
 - transactions, 93
- transactionmanager.cpp, 125
- transactionmanager.h, 125
- transactions
 - TransactionManager, 93
- Type
 - CompteHierarchie.h, 116
- TYPE_EXC_H
 - ExceptionHierarchie, 51
- UFND_EXC_F
 - ExceptionFichier, 50
- UKWN_EXC_C
 - ExceptionComptabilite, 48
- UKWN_EXC_F
 - ExceptionFichier, 50
- UKWN_EXC_H
 - ExceptionHierarchie, 51
- UKWN_EXC_T
 - ExceptionTransaction, 52
- undo
 - CompteHierarchie, 17
 - CompteReel, 35
 - CompteVirtuel, 42
- valide
 - Transaction, 82
- VIRTUEL
 - CompteHierarchie.h, 116
- visiter
 - Visiteur, 94, 95
 - VisiteurAffichage, 97
 - VisiteurFree, 100
 - VisiteurGetSolde, 103
 - VisiteurPere, 106
 - VisiteurPoste, 109
 - VisiteurRecherche, 112
- Visiteur, 94
 - visiter, 94, 95
- Visiteur.cpp, 126
- Visiteur.h, 126
- VisiteurAffichage, 95
 - afficherLigne, 96
 - CompteManager, 98
 - indentation, 98
 - operator=, 97
 - visiter, 97
 - VisiteurAffichage, 96
- VisiteurAffichage.cpp, 126
- VisiteurAffichage.h, 126
- VisiteurFree, 98
 - CompteHierarchie, 18
 - CompteManager, 100
 - CompteReel, 36
 - CompteVirtuel, 42
 - operator=, 99
 - visiter, 100
 - VisiteurFree, 99
- VisiteurFree.cpp, 126
- VisiteurFree.h, 126
- VisiteurGetSolde, 101
 - CompteManager, 103
 - getSolde, 102
 - operator=, 102
 - solde, 103
 - visiter, 103
 - VisiteurGetSolde, 101, 102
- VisiteurGetSolde.cpp, 127
- VisiteurGetSolde.h, 127
- VisiteurPere, 104
 - CompteManager, 106
 - getResultat, 105
 - operator=, 105
 - recherche, 106

- resultat, [106](#)
- visiter, [106](#)
- VisiteurPere, [104](#), [105](#)
- VisiteurPere.cpp, [127](#)
- VisiteurPere.h, [127](#)
- VisiteurPoste, [107](#)
 - changePoste, [108](#)
 - CompteManager, [109](#)
 - getResultat, [108](#)
 - operator=, [108](#)
 - p, [109](#)
 - resultat, [110](#)
 - visiter, [109](#)
 - VisiteurPoste, [107](#), [108](#)
- VisiteurPoste.cpp, [127](#)
- VisiteurPoste.h, [128](#)
- VisiteurRecherche, [110](#)
 - CompteManager, [112](#)
 - getResultat, [111](#)
 - idcompte, [112](#)
 - operator=, [111](#)
 - resultat, [113](#)
 - visiter, [112](#)
 - VisiteurRecherche, [111](#)
- VisiteurRecherche.cpp, [128](#)
- VisiteurRecherche.h, [128](#)