Projet de Bases de Données :
Système d'information d'une clinique dentaire

Rapport

Xavier Milhaud, Basile Voisin

2 mai 2007

Ce document détaille les differentes étapes de la conception de notre base de données et explique les principaux choix qui ont dû être fait afin d'aboutir à notre modélisation. Il regroupe aussi la mise en place de l'application de gestion de notre base de données.

Table des matières

1 Dépendances fonctionnelles et premières modélisations

1.1 Dèpendances fonctionnelles explicites

La lecture de l'énoncé nous a permis d'extraire certaines dépendances fonctionnelles particulièrement explicites :

- Personnel : **NSR** $\longrightarrow$**NomPers, PrenomPers, TelPers**
  Et pour des catégories de personnel bien définies :
  - **NSR** $\longrightarrow$**ProfAdmin, SalFixe** pour le personnel administratif ;
  - **NSR** $\longrightarrow$**ProfMed, TxHor** pour les médecins ;
  - **NSR** $\longrightarrow$**ProfAux, TxHor** pour le personnel auxiliaire médical.

  Le fait que ces trois catégories de personnel aient certaines propriété en commun et d'autres spécifiques à une catégorie nous amene naturellement à créer un sous-type de personnel par catégorie dans le schéma entités/associations.
- Patients : **NSS** $\longrightarrow$**NomPat, PrenomPat, SexPat, BirthPat, TelPat**
- Adresses des patients et du personnel (nous avons décomposé l'adresse de façon a n'avoir que des propriétés atomiques) :
  - **NSR** $\longrightarrow$**Num, Rue, CodePostal, Ville, Pays**
  - **NSS** $\longrightarrow$**Num, Rue, CodePostal, Ville, Pays**

  Ce qui correspond à l'entité *adresse*, et aux associations *persHabite*, et *patHabite*.
- Interventions : **NomInterv** $\longrightarrow$**Materiaux, CoutInterv, MontRemb, Duree** ce qui se traduit par l'entité *catalogue int* sur notre schéma.
- Fiche de soin / de consultation (il n'y a qu'une consultation par jour pour un patient donné) : **NSS, DateCons** $\longrightarrow$**CoutTot** Cette dépendace traduisant bien le fait qu'il n'y a pas de consultations sans patient nous conduit à modéliser l'entité *fiche soin* par une entité faible reliée à *patient* par l'association *subit* (coût tôtal d'une consultation : *cf.* section1.3).
- Factures : **NumFact** $\longrightarrow$**MontFact, DateButFact** (montant d'une facture : *cf.* section1.3).
- Mensualités : **NumFact, DateButMens** $\longrightarrow$**, MontMens** Il est clair ici qu'il ne peut y avoir de mensualité sans facture, c'est pourquoi nous représentons l'entité *mensualité* comme entité faible reliée à *facture* par l'association *divisee* (date butoir d'une mensualité : *cf.* section1.3).
- Rappel : **NumRappel** $\longrightarrow$**TypeRappel** d'où l'entité *rappel* (montant d'un rappel : *cf.* section1.3).
- Paiements : **NumPaiemt** $\longrightarrow$**DatePaiemt, MontantPaiemt** d'où l'entité *paiement* (montant d'un paiement : *cf.* section1.3).

1.2 Autres dépendances fonctionnelles

Aprés une analyse plus minutieuse du sujet, nous avons pu déterminer d'autres dépendaces fonctionnelles importantes :

- La fiche de soins est rédigée par un seul auxiliaire médical : **NSS, DateCons** $\longrightarrow$**NSR**, où NSR est le numéro d'un auxiliaire médical, ce que nous avons traduit par l'association *redige*.
- Une intervention est effectuée dans le cadre d'une consultation par un seul médecin (à l'aide de l'auxiliaire médical de la consultation) sur un seul patient et peut donner lieu à une observation : **NSS, DateCons, NSR, NomInterv** $\longrightarrow$**Observation**, où NSR est le numéro d'un médecin, ce qui est représenté par l'association ternaire *intervention*.
- Une consultation donne lieu à une et une seule facture : **NSS, DateCons** $\longrightarrow$**NumFac** et **NumFact** $\longrightarrow$**NSS, DateCons**. Nous aurions donc pu regrouper les entités *fiche soin* et *facture*, mais nous avons préféré nous contenter de les relier grâce à l'association *donne lieu* afin de séparer les informations administratives et médicales.
- Pour exprimer l'expérience d'un médecin : **NSR, NomInterv** $\longrightarrow$**NbExec** où NSR est le numéro d'un médecin, ce qui donne l'association *experience*.
- Une facture est réglée par un patient : **NumFact** $\longrightarrow$**NSS**, d'où l'association *regleFact*.
- Un paiement est effectué par un patient : **NumPaiemt** $\longrightarrow$**NSS**, d'où l'association *reglePaiemt*.
- Un paiement paie une ou plusieurs mensualités d'une ou plusieurs factures : **NumFact, DateButMens** $\longrightarrow$**NumPaiemt**, c'est ce que traduit l'association paie.

- Un rappel porte sur une mensualité d'une facture : **NumRappel** ⟶**DateButMens, NumFact**, ce qui est représenté par l'association *vise*.
- Un rappel définitif implique un paiement : **NumRappel** ⟶**NumPaiemt**, symbolisé par l'association *implique*.

1.3 Contraintes et remarques

1.3.1 Personnel

- Le calcul du salaire des médecins et du personnel auxiliaire médical se fait à partir de leur taux horaire, et du nombre d'heures qu'ils ont travaillé dans le mois. Ce nombre d'heures est obtenu
  - en sommant les durées des interventions effectuées pour les médecins,
  - en sommant les durées de toutes les interventions constituant les consultations dont ils ont tenu la fiche de soin pour le presonnel auxiliaire médical.
- L'acces aux différentes informations concernant un patient est limité en fonction de la catégorie de personnel voulant y accéder.
- Le salaire, fixe ou calculé, est bien sûr positif.

1.3.2 Patients

- A chaque patient peut être associé un profil : une ou plusieurs
  - allergies (entité *allergie* - association *souffre*)
  - contre-indications (entité *CI* - association *a pour*)
  - prothèses (entité *prothese* - association *a une proth*)
  - maladies héréditaires (entité *hereditaire* - association *atteint de*)

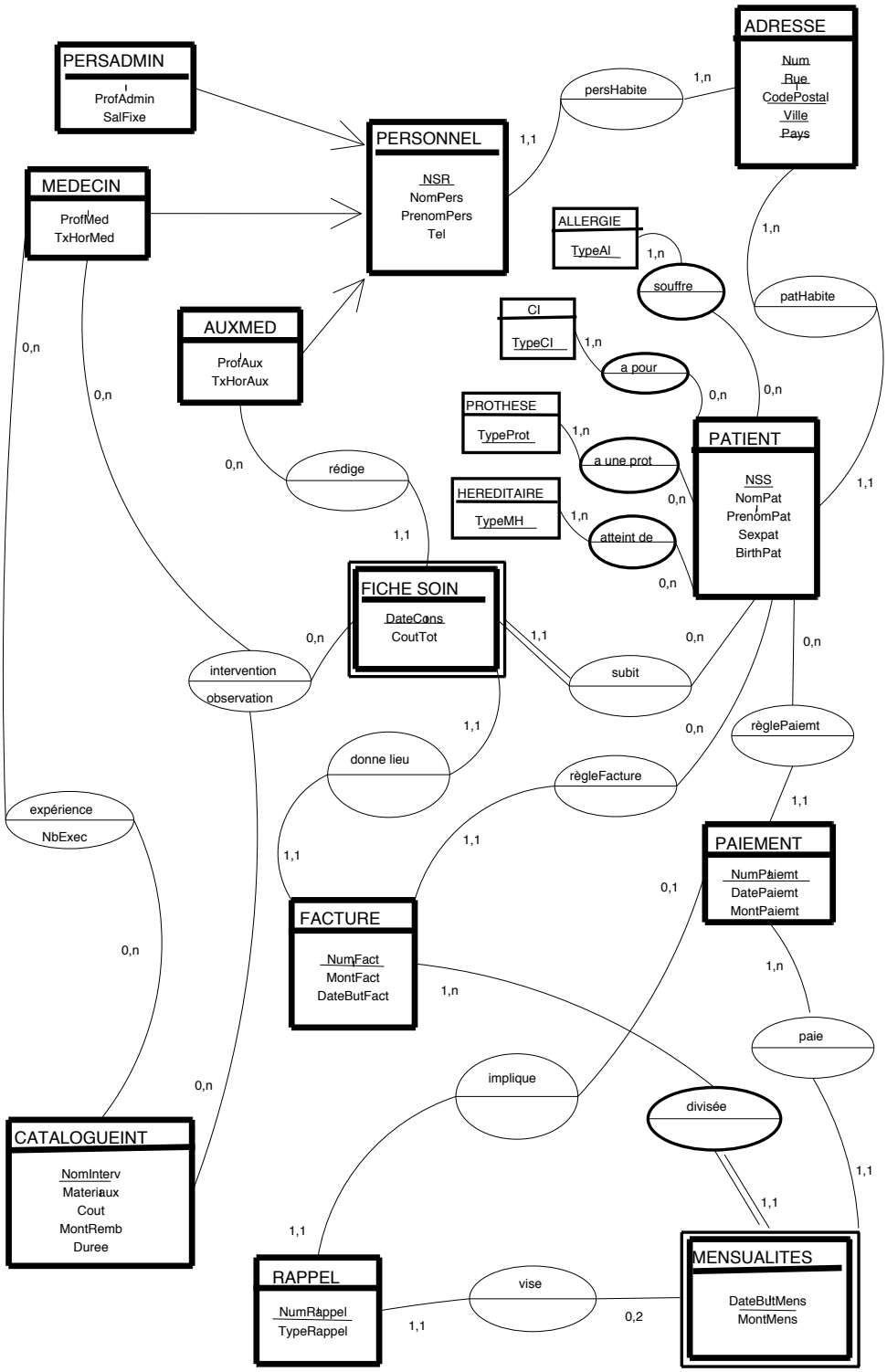
1.3.3 Soins

- La fiche de consultation est assimilée à la fiche de soin.
- Le nombre d'exécutions d'une intervention par un médecin est un entier positif ou nul.
- Le coût d'une intervention est positif.
- Le montant rembourse pour une intervention est positif et doit être inférieur au cout de cette intervention.
- Le coût total d'une consultation est positif et doit être supérieur ou égal à la somme des coûts des intreventions qu'elle regroupe auxquels on a soustrait le montant remboursé. Il est ainsi possible d'ajouter certains frais à une consultation et surtout de prendre en compte une consultation sans intervention.

1.3.4 Paiements

- Le montant d'une facture est positif et doit être égal au coût total de la consultation à laquelle elle est associée.
- Une facture donne toujours lieu à au moins une mensualité.
- Le montant des mensualités est positif (la clinique n'est pas une banque) et la somme des montants des mensualites d'une facture doit être égale au montant de la facture.
- La date butoir de la mensualité numéro i doit valoir $DateButFact - (i-1)mois$.
- Le type d'un rappel est soit 1 (rappel informatif) soit 2 (rappel définitif).
- Un rappel définitif ne peut avoir lieu que si un rappel informatif a déjà été fait.
- Le montant d'un paiement est positif et doit être égal à la somme des montants des mensualités qu'il règle éventuellement majorés du coût des rappels :
  - 12€
  - 12€+5% du montant de la mensualité (on suppose que le deuxième rappel nécessite également une lettre recommandée pour informer le patient du débit forcé)
- Etant donné qu'un rappel définitif implique un paiement, ce paiement en question ne devrait régler que la mensualité visée par ce rappel.

2 Schéma Entités/Associations



3 Passage au relationnel

Pour construire le schéma relationnel, nous avons appliqué les règles vues en cours.

3.1 Entités simples

- Etant donné que {PERSADMIN, MEDECIN, AUXMED} forme une partition du PERSONNEL, nous avons choisi de créer les relations suivantes de façon à éviter de stocker deux fois les mêmes informations (première solution du cours) :
  - **PERSADMIN (NSR, NomPers, PrenomPers, TelPers, ProfAdmin, SalFixe)**
  - **MEDECIN (NSR, NomPers, PrenomPers, TelPers, ProfMed, TxHorMed)**
  - **AUXMED (NSR, NomPers, PrenomPers, TelPers, ProfAux, TxHorAux)**
- **PATIENT (NSS, NomPat, PrenomPat, SexPat, BirthPat, TelPat)**
- **ADRESSE (Num, Rue, CodePostal, Ville, Pays)**
- **ALLERGIE (TypeAl)**
- **CI (TypeCI)**
- **PROTHESE (TypeProt)**
- **HEREDITAIRE (TypeMH)**
- **PAIEMENT (NumPaiemt, DatePaiemt, MontPaiemt)**
- **FACTURE (NumFact, MontFact, DateButFact)**
- **RAPPEL (NumRappel, TypeRappel)**
- **CATALOGUEINT (NomInterv, Materiaux, CoutInterv, MontRemb, Duree)**

3.2 Entités faibles

Notre schéma comporte deux entités faibles :
– **FICHESOIN (NSS, DateCons, CoutTot)**
– **MENSUALITE (NumFact, DateButMens, MontMens)**

3.3 Associations dont une des cardinalités est (1,1)

L'examen de ces associations nous a conduit aux modifications suivantes :
- association *persHabite* :
  - **PERSADMIN (NSR, NomPers, PrenomPers, TelPers, ProfAdmin, SalFixe, Num, Rue, CodePostal, Ville, Pays)**
  - **MEDECIN (NSR, NomPers, PrenomPers, TelPers, ProfMed, TxHorMed, NbHMed, Num, Rue, CodePostal, Ville, Pays)**
  - **AUXMED (NSR, NomPers, PrenomPers, TelPers, ProfAux, TxHorAux, NgHAux, Num, Rue, CodePostal, Ville, Pays)**
- association *patHabite* : **PATIENT (NSS, NomPat, PrenomPat, SexPat, BirthPat, TelPat, Num, Rue, CodePostal, Ville, Pays)**
- Les deux derniers points ainsi que le fait que toutes les propriétés de la relation **ADRESSE** fassent partie de la clef rendent la conservation de cette relation qui sera donc supprimée pour ne pas stocker d'information redondante.
- association *donne lieu* : nous aurions pu décider de rassembler les entités *fiche soin* et *facture* grâce à cette association qui est en fait une bijection, mais afin de facilité les accès restreints aux données nous avons décidé de simplement rajouter le numéro de la facture dans la fiche de soins : **FICHESOIN (NSS, DateCons, CoutTot, NumFact)** (*cf.* section3.6).
- association *redige* : **FICHESOIN (NSS, DateCons, CoutTot, NumFact, NSR)** (*cf.* section3.6).
- association *regleFacture* : **FACTURE (NumFact, MontFact, DateButFact, NSS)** (*cf.* section3.6).
- association *reglePaiemt* : **PAIEMENT (NumPaiemt, DatePaiemt, MontPaiemt, NSS)** (*cf.* section3.6).
- association *paie* : **MENSUALITE (NumFact, DateButMens, MontMens, NumPaiemt)**
- association *implique* : **RAPPEL (NumRappel, TypeRappel, NumPaiemt)**

- association *vise* : **RAPPEL (NumRappel, TypeRappel, NumPaiemt, NumFact, DateButMens)** (*cf.* section3.6).

3.4 Associations dont une des cardinalités est (0,1)

Nous n'avons pas de telles associations.

3.5 Autres associations

- association *experience* : **EXPERIENCE (NSR, NomInterv, NbExec)**
- association *intervention* : **INTERVENTION (NSR, DateCons, NSS, NomInterv, Observations)**
- association *SOUFFRE* : **SOUFFRE (NSS, TypeAl)**
- association *APOUR* : **APOUR (NSS, TypeCI)**
- association *AUNEPROTH* : **AUNEPROTH (NSS, TypeProt)**
- association *ATTEINT* : **ATTEINT (NSS, TypeMH)**

3.6 Contraintes supplémentaires introduites

- Dans la relation **FICHESOIN**, NSS doit correspondre à un des numéros de la relation **PATIENT**.
- Dans la relation **MENSUALITE**, NumFact doit correspondre à un des numéros de la relation **FACTURE**.
- Dans la relation **FICHESOIN**, tous les numéros de factures doivent être différents.
- Dans la relation **FICHESOIN**, NSR doit correspondre à un des numéros de la relation **PERSADMIN**.
- Dans les relations **regleFacture** et **ReglePaiement**, NSS doit correspondre à un des numéros de la relation **PATIENT**.
- Dans la relation **RAPPEL**, NumFact et DateButMens doivent correspondre à un numéros et une mensualité de la relation **MENSUALITE**.
- Dans la relation **EXPERIENCE**, NSR doit correspondre à un des numéros de la relation **MEDECIN**, et NomInterv doit correspondre à un des noms d'intervention de la relation **CATALOGUEINT**.
- Dans la relation **INTERVENTION**, NSR doit correspondre à un des numéros de la relation **MEDECIN**, DateCons à une date de la relation **FICHESOIN**, NSS à un numéro de la relation **PATIENT** et NomInterv à une intervention de la relation **CATALOGUEINT**.
- Dans les relations *SOUFFRE*, *APOUR*, *AUNEPROTH*, *ATTEINT*, NSS doit correspondre à un numéro de patient de la relation **PATIENT** et les Types doivent correspondre aux types contenus dans les relations **ALLERGIE**, **CI**,**PROTHESE** et **HEREDITAIRE**.

3.7 Commentaire

Lors de la création du schéma entités/associations puis lors du passage au relationnel, nous avons décidé de ne pas traduire directement certaines dépendances plus ou moins triviales. En effet, la traduction de ces dépendances aurait surchargé le schéma relationnel, et nous aurions perdu en place mémoire et en informations redondantes ce que nous aurions gagné en facilité d'écriture des requêtes SQL. Cependant, lors de l'écriture de ces requêtes, nous nous réservons le droit d'adapter la structure de notre base de données si cela s'avérait judicieux.

3.8 Bilan relationnel, formes normales et contraintes ne pouvant pas être vérifiées au niveau relationnel

3.8.1 Récapitulatif des relations et formes normales

RELATION	attributs	FN
PERSADMIN	(NSR, NomPers, PrenomPers, TelPers, ProfAdmin, SalFixe, Num, Rue, CodePostal, Ville, Pays)	3
MEDECIN	(NSR, NomPers, PrenomPers, TelPers, ProfMed, TxHorMed, Num, Rue, CodePostal, Ville, Pays)	3
AUXMED	(NSR, NomPers, PrenomPers, TelPers, ProfAux, TxHorAux, Num, Rue, CodePostal, Ville, Pays)	3
PATIENT	(NSS, NomPat, PrenomPat, TelPat, SexPat, BirthPat, Num, Rue, CodePostal, Ville, Pays)	3
ALLERGIE	(TypeAl)	3
CI	(TypeCI)	3
PROTHESE	(TypeProt)	3
HERED.	(TypeMH)	3
SOUFFRE	(NSS, TypeAl)	3
APOUR	(NSS, TypeCI)	3
AUNEPROTH	(NSS, TypeProt)	3
ATTEINT	(NSS, TypeMH)	3
PAIEMENT	(NumPaiemt, DatePaiemt, MontPaiemt, NSS)	2
FACTURE	(NumFact, MontFact, DateButFact, NSS)	2
RAPPEL	(NumRappel, TypeRappel, NumPaiemt, NumFact, DateButMens)	3
CATAL.INT	(NomInterv, Materiaux, CoutInterv, MontRemb, Duree)	3
FICHESOIN	(NSS, DateCons, CoutTot, NumFact, NSR)	3
MENSUALITE	(NumFact, DateButMens, MontMens, NumPaiemt)	3
EXPERIENCE	(NSR, NomInterv, NbExec)	3
INTERV.	(NSR, DateCons, NSS, NomInterv, Observations)	3

3.8.2 Contraintes à vérifier dans le programme

- Vu qu'il y a une relation pour le personnel administratif, une pour les auxiliaires medicaux et une autre pour les medecins, il faudra vérifier dans le programme que les NSR sont distincts.
- L'accés aux données est restreint en fonction de la catégorie de personnel.
- Une facture donne toujours lieu à au moins une mensualité.
- La date butoir de la mensualité numéro i doit valoir $DateButFact - (i-1)mois$.
- Un rappel définitif ne peut avoir lieu que si un rappel informatif a déjà été fait.
- Le montant d'un paiement est positif et doit être égal à la somme des montants des mensualités qu'il règle éventuellement majorés du coût des rappels :
  - 12€
  - 12€+5% du montant de la mensualité
- Etant donné qu'un rappel définitif implique un paiement, ce paiement en question ne devrait régler que la mensualité visée par ce rappel.
- Le coût total d'une consultation doit être supérieur ou égal à la somme des coûts des intreventions qu'elle regroupe. Il est ainsi possible d'ajouter certains frais à une consultation et surtout de prendre en compte une consultation sans intervention.
- Le montant d'une facture doit être égal au coût total de la consultation à laquelle elle est associée moins le montant rembourse de chaque intervention de cette consultation.
- Le montant rembourse d'une intervention doit être inférieur ou égal au coût de cette intervention.

4 Fonctionnalités supportées

Les fonctionnalités supportées par notre base de données sont les suivantes :

4.1 Gestion des patients

- liste des patients (`NSS`, `Nom` et `Prénom`) soignés (`listePatient`) ;
- ajout d'un patient (`addPatient`) ;
- mise à jour des coordonnées d'un patient (et écrasement des anciennes coordonnées) (`setCoordPatient`) ;
- numéros de sécurité sociale correspondant à un nom donné (`getNSSPatient`) ;
- informations administratives sur un patient (`getInfoPatient`) ;
- ajout d'une allergie, contre-indication, prothèse et maladie héréditaire (`addAllergie`, `addCI`, `addProth`, `addHered`) ;
- mise à jour du profil médical d'un patient (on suppose qu'on ne peut pas enlever une allergie, prothèse etc. à un patient) (`addAllergiePatient`, `addCIPatient`, `addProthPatient`, `addHeredPatient`) ;
- accés au profil médical d'un patient (`getProfilPatient`).

4.2 Gestion du personnel

- liste du personnel (`NSR`, `Nom`, `Prénom`, `Profession`) (`listePers`) ;
- ajout d'un membre du personnel (`addAdmin`, `addAux`, `addMed`) ;
- mise à jour des coordonnées d'un membre du personnel (et écrasement des anciennes coordonnées) (`setCoordPers`) ;
- numéros de personnel correspondant à un nom donné (`getNSRPers`) ;
- informations administratives sur un membre du personnel (`getInfoPers`) ;
- modification du salaire (ou du taux horaire) d'un membre du personnel (`setSalPers`) ;
- calcul du salaire d'un mois donné pour un membre du personnel donné (`calcSalPers`) ;
- modification de la profession d'un membre du personnel (`setProfPers`) ;
- accés à l'expérience d'un médecin (`getExpMed`) ;
- ajout d'une intervention à l'expérience d'un médecin (`addExpMed`).

4.3 Gestion des soins

- ajout d'une consultation (qui donne lieu à une fiche de soin) (`addConsult`) ;
- détail d'une consultation donnée dont la liste des interventions (`getInfoConsult`),
- liste des interventions cataloguées (`getListeInterv`) ;
- ajout/modification d'une intervention dans le catalogue des interventions (`setInterv`) ;
- ajout d'une intervention dans le cadre d'une consultation (`addIntervConsult`).

4.4 Historique des soins

- liste de toutes les fiches de soin (`getListeConsult`),
- liste des consultations pour un patient donné (`getListeConsultPatient`) ;
- liste de toutes les interventions (`getListeInterv`),
- liste des interventions pour un patient donné (`getListeIntervPatient`) ;

4.5 Gestion des rappels de paiement

- liste des factures pour un patient donné (`getListeFactPatient`),
- définition des mensualités pour une facture donnée (`setMensFact`),
- detail des mensualites d'une facture donnée (`getMensFact`),
- liste des mensualites impayées pour un patient donné (`getMensImpPatient`) ;
- liste des mensualites impayées dont la date butoir est depassée pour un patient donné (`getMensDepPatient`) ;
- édition des rappels pour un patient donné (`setRappelPatient`) ;

- liste des rappels pour un patient donné (`getRapPatient`) ;
- enregistrement d'un paiement effectué par un patient (`setPaiemtPatient`) ;
- liste des paiements pour un patients donné (`listePaiemtPatient`).

4.6 Statistiques

- nombre moyen de consultations par jour (`moyConsultJour`) ;
- nombre moyen de consultations par mois (`moyConsultMois`) ;
- nombre de patients ayant des factures impayées (`nbPatientFactImp`) ;
- nombre de patients dans la clinique (`nbPatient`) ;
- effectif du personnel dans la clinique (`nbPers`).

5 Transactions et requêtes

5.1 Quand gère-t-on les transactions ?

Une transaction est utilisée lors d'une modification de la base de données, donc lors d'un ajout, d'une suppression ou d'une mise à jour de données dans la base.
Nous avons décidé de valider les transactions (grâce à la fonction `commit`) manuellement dans les méthodes, et nous mettons donc dans le programme principal la fonction de validation automatique de la transaction à faux (`setAutoCommit`).
Remarque : nous avons choisi de ne pas vous présenter dans ce rapport toutes les requêtes associées à chaque méthode pour la simple et bonne raison que un très grand nombre de requêtes sont similaires et cela aboutirait à un rapport surchargé qui contiendrait très souvent la même information. C'est pourquoi nous vous donnons dans la suite seulement la forme de ces requêtes. Ainsi, nous gérons les transactions dans différentes méthodes qui sont les suivantes :

- Toutes les méthodes d'ajout :
  - ajout de patient par la méthode `addPatient`,
  - ajout de nouvelles maladies dans les catalogues correspondants (allergie...) par les méthodes `addAllergie`, `addCI`, `addProt` et `addHered`,
  - ajout de problème de santé (allergie...) pour un patient donné par les méthodes `addAllergiePatient`, `addCIPatient`, `addProtPatient` et `addHeredPatient`,
  - ajout de personnel par la méthode `addAdmin`, `addMed` et `addAux`,
  - ajout de consultation (et génération de la facture) par la méthode `addCons`,
  - ajout d'intervention dans le catalogue par la méthode `addInterv`,
  - ajout d' expérience pour une intervention à un médecin par la méthode `addExpMed`,
  - ajout d'intervention dans le cadre d'une consultation (et mise à jour de l'expérience du médecin concerné) par la méthode `addIntervConsult`,
  - ajout de mensualités pour une facture par la méthode `setMensFact`,
  - ajout d'un paiement (et mise a jour des mensualités réglées) par la méthode `addPaiemtPatient`,
  - génération des rappels concernant les mensualités (et génération éventuelle de paiements) par la méthode `setRappelPatient`.

  La requête qui permet d'effectuer ces ajouts dans la base de données est la suivante :

  ```
  insert into table values (valeurs)
  ```

  De fait, cette requête SQL a des paramètres à définir qui sont en général les paramètres correspondant aux clefs des relations, ceci est fait via les méthodes `setInt`, `setFloat` et `setString`. Nous aurons donc une transaction à valider ou non à chaque fois que nous aurons cette requête. Ces transactions sont validées (`commit`) dans le cas où l'on insère un n-uplets dont la clef n'est pas déjà présente dans la base de données, donc dans le cas où les contraintes de clefs sont vérifiées.
  Sinon elle renvoie une exception de type SQL rattrapée proprement par notre programme qui affiche la raison de cette exception, et nous revenons à l'état antérieur de la base de données grâce à la méthode

`rollback`.

- Toutes les méthodes de suppression : par défaut on ne peut rien supprimer pour les raisons suivantes :
  - suppression d'un médecin ou d'un auxiliaire médical : impossible car il faut pouvoir garder l'historique d'une intervention par exemple avec donc le medecin qui l'aurait effectuée ou bien l'auxiliaire médical ayant participé à une consultation,
  - suppression d'un patient : impossible car il faut pouvoir garder les informations administratives ou médicales d'un patient en cas de demande d'informations par une autre clinique dans le futur,
  - suppression des consultations, interventions : impossible car il faut pouvoir remonter dans le passé pour savoir qui a fait quoi. Par exemple, le catalogue des interventiuons répertorie toutes les interventions : une fois qu'une intervention a été ajoutée il ne faut donc surtout pas pouvoir la supprimer.

  En résumé, les traces administratives sont primordiales pour le suivi du personnel et des patients, il est donc illogique de pouvoir supprimer ces traces !
  Cependant, la requête SQL permettant la suppression dans la base de données est la suivante :
  `delete from table where condition`
- Toutes les méthodes de mise à jour :
  - mise à jour des coordonnées d'un patient par la méthode `setCoordPatient`,
  - mise à jour des coordonnées d'un membre du personnel (administratif, auxiliaire médical ou médecin) par la méthode `setCoordPers`,
  - mise à jour du salaire ou taux horaire d'un membre du personnel par la méthode `setSalPers`,
  - modification de la profession d'un membre du personnel par la méthode `setProfPers`,
  - mise à jour des caractéristiques d'une intervention dans le catalogue des interventions par la méthode `setInterv`,
  - mise à jour de l'expérience médicale d'un médecin pour une consultation par la méthode `addExpMed`,
  - mise à jour des mensualités correspondantes à un paiement par la méthode `addPaiemtPatient`
  - mise à jour de la fiche de soin et de la facture après ajout d'une intervention à la consultaiton (montants) grâce à la méthode `addInterConsult`.
  - mise à jour des mensualitées réglées par un paiement par la fonction `addPaiemtPatient`.

  Toutes ces modifications de la base de données, lorsqu'elles sont effectuées, impliquent donc une transaction via la requête SQL suivante :
  `update table set attribut where condition`
  La validation de la transaction se fait dans le cas où il n'y a pas d'exception de type SQL et de contraintes violées via la méthode `commit`.
  En cas de modification d'une donnée inexistante dans la base, nous affichons le message correspondant et revenons dans l'état antérieur de la base grâce à la fonction `rollback` en fonction du résultat fourni par la méthode `executeUpdate`. En effet, si elle renvoie un entier qui vaut 1, cela signifie que la base de données a été modifiée et donc un `rollback` est nécessaire.

Remarque importante : les transactions sont gérées dans **toutes** les méthodes en cas d'echec. En effet, il existe des exceptions que nous nous attendons à avoir (ex : insertion dans la base de données d'un n-uplets dont la clef est déjà présente dans la base -> contrainte violée) et que nous rattrapons donc proprement, mais il existe également des exceptions que SQL peut lancer et auxquelles nous ne nous attendons pas : ces exceptions sont prises en compte dans toutes les méthodes et génèrent une annulation de la transaction par la méthode `rollback()` lorsqu'elles sont rattrapées (dans le catch). La violation de contraintes à vérifier dans le programme ne lève pas d'exception, cependant nous annulons la transaction et produisons un affichage adapté.

5.2 Vérification des transactions

Pour tester les fonctionnalités de notre programme, nous avons rempli la base de données à l'aide d'un script utilisant notre logiciel (cf. `fichiers.cli`) Les transactions sont vérifiées grâce à nos fonctions de consultation de la base de données.
En effet, après consultation, il est facile de voir si la modification a bien été faite ou non. La consultation

de la base de données n'induit donc aucune transaction puisque l'état de celle-ci reste inchangé. Toutes les consultations de la base de données se font grâce à la requête SQL

`select attribut(s) from table(s) where condition(s)`

Ceci est donc la forme générale d'une requête de consultation de la base, sachant que cette forme peut être beaucoup plus compliquée suivant ce que l'on veut consulter. Les fonctions de notre programme servant à cet effet sont de la forme `getNomFonction`.
Remarque : pour des raisons de clarté, nous ne mettons pas ici toutes les requêtes SQL de ce type puisque elles sont toutes de la même forme. Voici quelques exemples de requêtes SQL de notre programme :

- des fonctions de calcul peuvent être utilisées dans la clause select (ex : `count()`),
  select count(*)
  from PATIENT
- des conditions peuvent être compliquées,
- des clauses supplémentaires peuvent être utilisées (ex : `group by`),
- utilisation de requêtes imbriquées, par exemple pour récupérer le nombre moyen de consultations par jour :
  select count(*)/NbJourDiff
  from FICHESOIN
  where NbJourDiff in (select NbJourDiff = count(distinct DateCons) from FICHESOIN)

- des mots-clefs (ex : `distinct`) :
  select count(*)/nbMoisDiff
  from FICHESOIN
  where NbMoisDiff in (select NbMoisDiff = count(distinct trunc(DateCons,'MM')) from FICHESOIN)

6 Contraintes vérifiées dans notre programme

- Vérification de l'unicité des NSR :
  Nous vérifions "à la main" dans les méthodes d'ajout d'un membre du personnel que le nouveau NSR n'existe pas déjà dans les relations `PERSADMIN`, `AUXMED` et `MEDECIN`.
  Les méthodes `NSSestDsTable` et `NSRestDsTable` font ce contrôle grâce à une requête SQL ne modifiant pas la base de données (`select NSS from table where NSS =?`), pour laquelle le numéro du patient cherché (unique) est passé en paramètre de la méthode `NSSestDsTable`. Nous parcourons ensuite le résultat (n-uplet) qui doit être vide si l'on veut pouvoir ajouter ce membre.
- Restriction de l'accès aux données :
  Afin de restreindre la consultation et la modification de la base de données en fonction de la catégorie de personnel, nous avons prévu un système d'authentification de l'utilisateur. Par défaut, l'utilisateur est un membre du personnel administratif mais peut changer de statut dans le programme principal afin d'accéder aux différentes fonctionnalités du logiciel. Concrètement, la vérification de la catégorie du personnel se fait grâce à la fonction `authVerif` juste avant les appels de fonctions contenant les requetes SQL dans la classe `CliniqueDentaire`.
- Le coût total d'une consultation doit être supérieur ou égal à la sommme des coûts des interventions qui en font partie : la méthode `addIntervConsult` vérifie cette contrainte par construction grâce à l'instruction suivante :
  `setFloat(1,CoutCons+CoutInterv).`
  Nous mettons donc le montant à jour à chaque ajout.
- Une facture donne lieu à au moins une mensualité : contrainte vérifiée dans la méthode `setMensFact` de la classe de gestion des paiements. La variable `nbMens` est supérieure ou égale à 1.
- Le montant remboursé d'une intervention est inférieur ou égal au coût de cette intervention : vérifié dans le menu des consultations et plus particulièrement dans la méthode `menuConsult` de la classe `CLiniqueDentaire` grâce à un simple test.

- Date butoir des mensualités : gérée également dans la méthode `setMensFact`. Nous récupérons pour cela la date butoir de la facture à partir de laquelle avec le nombre de mensualités nous calculons la date butoir de chaque mensualité.
- Montant d'une facture égal au coût total de la consultation associée moins les remboursements : cette propriété se fait aussi par construction grâce à l'instruction suivante `setFloat(1,MontFact+CoutInterv-remb)` dans la méthode `addIntervConsult` de la classe de gestion des consultations.
- Le rappel définitif n'est émis que dans le cas où un rappel informatif a déjà été émis : contrainte garantie par la structure de la boucle while de la méthode `setRappelPatient`.

7 Détail des fonctions Java

Nous avons organisé notre code en une classe principale `CliniqueDentaire` regroupant les menus et l'interface, la gestion de l'authentification et effectuant les appels aux fonctions contenant les requêtes SQL. Ces fonctions sont naturellement réparties dans les 6 classes `GestionPatient`, `GestionPersonnel`, `GestionConsultation`, `Historique`, `GestionPaiement` et `Statistiques` qui reprenent les sous-menus du programme. Enfin, une classe `Parametres` regroupe la saisie propre et securitaire des parametres a passer aux fonctions gérant les transactions et ayant accés à la base de données.

Pour plus de détails, voir la `javadoc` en annexe.

Annexe 1 : script de la création de la base de données

Attention : ce script a été modifié très fortement par rapport à la version précédente qui contenait beaucoup d'erreurs à la compilation.

```
CREATE TABLE PERSADMIN
( NSR INTEGER CONSTRAINT KPERSADMIN PRIMARY KEY,
  NomPers VARCHAR(50) NOT NULL,
  PrenomPers VARCHAR(50) NOT NULL,
  TelPers VARCHAR(10) NOT NULL,
  ProfAdmin VARCHAR(50) NOT NULL,
  SalFixe DECIMAL(10,2) NOT NULL CONSTRAINT CPERSADMINSAL CHECK (SalFixe > 0),
  Num INTEGER NOT NULL CONSTRAINT PERSADMINNUM CHECK (Num > 0),
  Rue VARCHAR(50) NOT NULL,
  CodePostal DECIMAL(5,0)  NOT NULL CONSTRAINT CPERSADMINCDEPOS CHECK (CodePostal > 0),
  Ville VARCHAR(50) NOT NULL,
  Pays VARCHAR(50) NOT NULL);

CREATE TABLE AUXMED
( NSR INTEGER CONSTRAINT KAUXMED PRIMARY KEY,
  NomPers VARCHAR(50) NOT NULL,
  PrenomPers VARCHAR(50) NOT NULL,
  TelPers VARCHAR(10) NOT NULL,
  ProfAux VARCHAR(50) NOT NULL,
  TxHorAux DECIMAL(6,2) NOT NULL CONSTRAINT CAUXMEDTX CHECK (TxHorAux > 0),
  Num INTEGER  NOT NULL CONSTRAINT CAUXMEDNUM CHECK (Num > 0),
  Rue VARCHAR(50) NOT NULL,
  CodePostal DECIMAL(5,0)  NOT NULL CONSTRAINT CAUXMEDCDEPOS CHECK (CodePostal > 0),
  Ville VARCHAR(50) NOT NULL,
  Pays VARCHAR(50) NOT NULL) ;

CREATE TABLE MEDECIN
( NSR INTEGER CONSTRAINT KMEDECIN PRIMARY KEY,
  NomPers VARCHAR(50) NOT NULL,
  PrenomPers VARCHAR(50) NOT NULL,
  TelPers VARCHAR(10) NOT NULL,
  ProfMed VARCHAR(50) NOT NULL,
  TxHorMed DECIMAL(6,2) NOT NULL CONSTRAINT CMEDECINTX CHECK (TxHorMed > 0),
  Num INTEGER NOT NULL CONSTRAINT CMEDECINNUM CHECK (Num > 0),
  Rue VARCHAR(50) NOT NULL,
  CodePostal DECIMAL(5,0) NOT NULL CONSTRAINT CMEDECINCDEPOS CHECK (CodePostal > 0),
  Ville VARCHAR(50) NOT NULL,
  Pays VARCHAR(50) NOT NULL) ;

CREATE TABLE PATIENT
( NSS INTEGER CONSTRAINT KPATIENT PRIMARY KEY,
  NomPat VARCHAR(50) NOT NULL,
  PrenomPat VARCHAR(50) NOT NULL,
  TelPat VARCHAR(10) NOT NULL,
  SexPat CHAR(1) NOT NULL CONSTRAINT CPATIENTSEX CHECK (SexPat IN ('F', 'M')),
  BirthPat DATE NOT NULL,
  Num INTEGER NOT NULL CONSTRAINT CPATIENTNUM CHECK (Num > 0),
  Rue VARCHAR(50) NOT NULL,
```

```
  CodePostal DECIMAL(5,0) NOT NULL CONSTRAINT CPATIENTCDEPOS CHECK (CodePostal > 0),
  Ville VARCHAR(50) NOT NULL,
  Pays VARCHAR(50) NOT NULL) ;

CREATE TABLE ALLERGIE
( TypeAl VARCHAR(50) CONSTRAINT KALLERGIE PRIMARY KEY) ;

CREATE TABLE SOUFFRE
( TypeAl VARCHAR(50),
  NSS INTEGER,
  CONSTRAINT FKSOUFFRE1 FOREIGN KEY (TypeAl) REFERENCES ALLERGIE(TypeAL),
   CONSTRAINT FKSOUFFRE2 FOREIGN KEY (NSS) REFERENCES PATIENT(NSS),
  CONSTRAINT KSOUFFRE PRIMARY KEY (TypeAl, NSS)) ;

CREATE TABLE CI
( TypeCI VARCHAR(50) CONSTRAINT KCI PRIMARY KEY) ;

CREATE TABLE APOUR
( TypeCI VARCHAR(50),
  NSS INTEGER,
  CONSTRAINT FKAPOUR1 FOREIGN KEY (TypeCI) REFERENCES CI(TypeCI),
  CONSTRAINT FKAPOUR2 FOREIGN KEY (NSS) REFERENCES PATIENT(NSS),
  CONSTRAINT KAPOUR PRIMARY KEY (TypeCI, NSS)) ;

CREATE TABLE PROTHESE
( TypeProt VARCHAR(50) CONSTRAINT KPROTHESE PRIMARY KEY) ;

CREATE TABLE AUNEPROTH
( TypeProt VARCHAR(50),
  NSS INTEGER,
  CONSTRAINT FKAUNEPROTH1 FOREIGN KEY (TypeProt) REFERENCES PROTHESE(TypeProt),
  CONSTRAINT FKAUNEPROTH2 FOREIGN KEY (NSS) REFERENCES PATIENT(NSS),
  CONSTRAINT KAUNEPROTH PRIMARY KEY (TypeProt, NSS)) ;

CREATE TABLE HEREDITAIRE
( TypeMH VARCHAR(50) CONSTRAINT KHEREDITAIRE PRIMARY KEY) ;

CREATE TABLE ATTEINT
( TypeMH VARCHAR(50),
  NSS INTEGER,
  CONSTRAINT FKATTEINT1 FOREIGN KEY (TypeMH) REFERENCES HEREDITAIRE(TypeMH),
  CONSTRAINT FKATTEINT2 FOREIGN KEY (NSS) REFERENCES PATIENT(NSS),
  CONSTRAINT KATTEINT PRIMARY KEY (TypeMH, NSS)) ;

CREATE TABLE CATALOGUEINT
( NomInterv VARCHAR(50) CONSTRAINT KCATALOGUEINT PRIMARY KEY,
  Materiaux VARCHAR(50) NOT NULL,
  CoutInterv DECIMAL(6,2) NOT NULL CONSTRAINT CCATALOGUEINTCOUT CHECK (CoutInterv > 0),
  MontRemb DECIMAL(6,2) NOT NULL
   CONSTRAINT CCATALOGUEINTREMB1 CHECK (0 <= MontRemb),
  Duree INTEGER NOT NULL CONSTRAINT CCATALOGUEINTDUR CHECK (Duree > 0),
  CONSTRAINT UCATALOGUEINT UNIQUE (Materiaux, CoutInterv, MontRemb, Duree)) ;
-- duree en minutes car TIME plante
```

```
CREATE TABLE FICHESOIN ( NSS INTEGER, DateCons DATE NOT NULL, CoutTot DECIMAL(6,2) NOT NULL CONSTRAINT

CREATE TABLE FACTURE ( NumFact INTEGER CONSTRAINT KFACTURE PRIMARY KEY CONSTRAINT CFACTURENUM CHECK (Nu

CREATE TABLE MENSUALITE
( NumFact INTEGER,
  DateButMens DATE,
  MontMens DECIMAL(6,2) NOT NULL CONSTRAINT CMENSUALITEMONT CHECK (MontMens > 0),
  NumPaiemt INTEGER,
  CONSTRAINT KMENSUALITE PRIMARY KEY (NumFact, DateButMens),
  CONSTRAINT FKMENSUALITE1 FOREIGN KEY (NumFact) REFERENCES FACTURE(NumFact)) ;

CREATE TABLE PAIEMENT
( NumPaiemt INTEGER
     CONSTRAINT KPAIEMENT PRIMARY KEY
 CONSTRAINT CPAIEMENT CHECK (NumPaiemt > 0),
  DatePaiemt DATE NOT NULL,
  MontPaiemt DECIMAL(6,2) NOT NULL CONSTRAINT CPAIEMENTMONT CHECK (MontPaiemt > 0),
  NSS INTEGER,
  CONSTRAINT FKPAIEMENT FOREIGN KEY (NSS) REFERENCES PATIENT(NSS)) ;

CREATE TABLE RAPPEL
( NumRappel INTEGER
CONSTRAINT KRAPPEL PRIMARY KEY
CONSTRAINT CRAPPELNUM CHECK (NumRappel > 0),
  TypeRappel INTEGER NOT NULL CONSTRAINT CRAPPELTYPE CHECK (TypeRappel IN (1, 2)),
  NumPaiemt INTEGER,
  NumFact INTEGER,
  DateButMens DATE,
  CONSTRAINT FKRAPPEL1 FOREIGN KEY (NumPaiemt) REFERENCES PAIEMENT(NumPaiemt),
  CONSTRAINT FKRAPPEL2 FOREIGN KEY (NumFact, DateButMens) REFERENCES MENSUALITE(NumFact, DateButMens))

CREATE TABLE EXPERIENCE
( NSR INTEGER,
  NomInterv VARCHAR(50),
  NbExec INTEGER CONSTRAINT CEXPERIENCENB CHECK (NbExec >= 0),
  CONSTRAINT KEXPERIENCE PRIMARY KEY (NSR, NomInterv),
  CONSTRAINT FKEXPERIENCE1 FOREIGN KEY (NSR) REFERENCES MEDECIN(NSR),
  CONSTRAINT FKEXPERIENCE2 FOREIGN KEY (NomInterv) REFERENCES CATALOGUEINT(NomInterv)) ;

CREATE TABLE INTERVENTION
( NSR INTEGER,
  DateCons DATE,
  NSS INTEGER,
  NomInterv VARCHAR(50),
  Observations VARCHAR(1000),
  CONSTRAINT KINTERVENTION PRIMARY KEY (NSR, DateCons, NSS, NomInterv),
  CONSTRAINT FKINTERVENTION3 FOREIGN KEY (NSS, DateCons) REFERENCES FICHESOIN(NSS, DateCons),
  CONSTRAINT FKINTERVENTION1 FOREIGN KEY (NSR) REFERENCES MEDECIN(NSR),
  CONSTRAINT FKINTERVENTION2 FOREIGN KEY (NomInterv) REFERENCES CATALOGUEINT(NomInterv)) ;
```

Annexe 2 : Javadoc

Class CliniqueDentaire

```
java.lang.Object
  └CliniqueDentaire
```

```
public class CliniqueDentaire
extends java.lang.Object
```

classe de gestion d'une clinique dentaire

Field Summary

private int	**choix**
private java.sql.Connection	**conn**
(package private) static java.lang.String	**CONN_URL**
(package private) static java.lang.String	**PASSWD**
(package private) static java.lang.String	**USER**
private int	**whoami**

Constructor Summary

CliniqueDentaire() constructeur

Method Summary

(package private) static void	**authReq**(int ad, int med, int aux) message lorsque l'authentification n'est pas bonne pour la requete demandee
(package private) int	**authVerif**(int ad, int med, int aux) verification de l'authentification pour la requete demandee

static void	**main**(java.lang.String[] arg) programme principal public static void main(String[] arg){ CliniqueDentaire clinique = new CliniqueDentaire() ; }
void	**mainMenu**() interface : menu principal du programme
(package private) void	**menuAddPers**(GestionPersonnel gp) ajout d'un membre du personnel
(package private) void	**menuAlCIPrMH**(GestionPatient gp, int code) ajout d'une allergie, contre-indication etc.
(package private) void	**menuAuth**() authentification
(package private) void	**menuConsult**() gestion des consultations
(package private) void	**menuHist**() historique des soins pour un patient
(package private) void	**menuListePers**(GestionPersonnel gp) liste du personnel
(package private) void	**menuPat**() gestion des patients
(package private) void	**menuPers**() gestion du personnel
(package private) void	**menuRappel**() gestion des rappels de paiement
(package private) void	**menuRmPers**(GestionPersonnel gp) suppression d'un membre du personnel
(package private) void	**menuSalPers**(GestionPersonnel gp) mise a jour du salaire d'un membre du personnel
(package private) void	**menuStat**() statistiques
(package private) void	**reset**() fonction de formatage de la base de donnees

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

CONN_URL

```
static final java.lang.String CONN_URL
```

See Also:
Constant Field Values

USER

```
static final java.lang.String USER
```

See Also:
Constant Field Values

PASSWD

```
static final java.lang.String PASSWD
```

See Also:
Constant Field Values

---

conn

```
private java.sql.Connection conn
```

---

whoami

```
private int whoami
```

---

choix

```
private int choix
```

Constructor Detail

CliniqueDentaire

```
public CliniqueDentaire()
```

constructeur

Method Detail

mainMenu

```
public final void mainMenu()
                   throws java.sql.SQLException
```

interface : menu principal du programme

Throws:
`java.sql.SQLException`

---

menuPat

```
final void menuPat()
            throws java.sql.SQLException
```

gestion des patients

Throws:
`java.sql.SQLException`

---

menuAlCIPrMH

```
final void menuAlCIPrMH(GestionPatient gp,
                        int code)
                 throws java.sql.SQLException
```

ajout d'une allergie, contre-indication etc.

Parameters:
`gp` - objet GestionPatient permettant d'appeller les fonctions refermant les requetes
`code` - code permettant d'ajouter l'allergie etc. dans le catalogue (0) ou a un patient (!= 0)
Throws:
`java.sql.SQLException`

---

menuPers

```
final void menuPers()
             throws java.sql.SQLException
```

gestion du personnel

Throws:
java.sql.SQLException

---

menuListePers

```
final void menuListePers(GestionPersonnel gp)
                  throws java.sql.SQLException
```

liste du personnel

Parameters:
gp - objet GestionPersonnel permettant d'appeller les fonctions refermant les requetes
Throws:
java.sql.SQLException

---

menuAddPers

```
final void menuAddPers(GestionPersonnel gp)
                throws java.sql.SQLException
```

ajout d'un membre du personnel

Parameters:
gp - objet GestionPersonnel permettant d'appeller les fonctions refermant les requetes
Throws:
java.sql.SQLException

---

menuRmPers

```
final void menuRmPers(GestionPersonnel gp)
               throws java.sql.SQLException
```

suppression d'un membre du personnel

Parameters:
gp - objet GestionPersonnel permettant d'appeller les fonctions refermant les requetes
Throws:
java.sql.SQLException

---

menuSalPers

```
final void menuSalPers(GestionPersonnel gp)
                throws java.sql.SQLException
```

mise a jour du salaire d'un membre du personnel

Parameters:
`gp` - objet GestionPersonnel permettant d'appeller les fonctions refermant les requetes
Throws:
`java.sql.SQLException`

---

menuConsult

```
final void menuConsult()
               throws java.sql.SQLException
```

gestion des consultations

Throws:
`java.sql.SQLException`

---

menuHist

```
final void menuHist()
             throws java.sql.SQLException
```

historique des soins pour un patient

Throws:
`java.sql.SQLException`

---

menuRappel

```
final void menuRappel()
              throws java.sql.SQLException
```

gestion des rappels de paiement

Throws:
`java.sql.SQLException`

---

menuStat

```
final void menuStat()
             throws java.sql.SQLException
```

statistiques

Throws:
`java.sql.SQLException`

---

menuAuth

```
final void menuAuth()
```

authentification

---

authVerif

```
final int authVerif(int ad,
                    int med,
                    int aux)
```

verification de l'authentification pour la requete demandee

Parameters:

`ad` - different de 0 si le presonnel administratif est autorise
`med` - different de 0 si les medecins sont autorises
`aux` - different de 0 si les auxiliaires medicaux sont autorises

Returns:

0 si l'authentification est correcte, != 0 sinon

---

authReq

```
static final void authReq(int ad,
                          int med,
                          int aux)
```

message lorsque l'authentification n'est pas bonne pour la requete demandee

Parameters:

`ad` - different de 0 si le presonnel administratif est autorise
`med` - different de 0 si les medecins sont autorises
`aux` - different de 0 si les auxiliaires medicaux sont autorises

---

reset

```
final void reset()
          throws java.sql.SQLException
```

fonction de formatage de la base de donnees

Throws:

`java.sql.SQLException`

---

main

```
public static void main(java.lang.String[] arg)
```

programme principal public static void main(String[] arg){ CliniqueDentaire clinique = new CliniqueDentaire() ; }

Package Class **Tree** **Deprecated** **Index** **Help**

PREV CLASS **NEXT CLASS** **FRAMES** **NO FRAMES** **All Classes**
SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

Class Coordonnees

```
java.lang.Object
  └Coordonnees
```

```
public final class Coordonnees
extends java.lang.Object
```

classe de manipulation des coordonnes d'un patient

Field Summary

private int	**code**
private int	**num**
private java.lang.String	**pays**
private java.lang.String	**rue**
private java.lang.String	**tel**
private java.lang.String	**ville**

Constructor Summary

Coordonnees(java.lang.String tel, int num, java.lang.String rue, int code, java.lang.String ville, java.lang.String pays) constructeur

Method Summary

int	**getCode**() accesseur code postal
int	**getNum**() accesseur numero de rue

java.lang.String	**getPays**() accesseur pays
java.lang.String	**getRue**() accesseur nom de rue
java.lang.String	**getTel**() accesseur telephone
java.lang.String	**getVille**() accesseur ville

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

tel

```
private java.lang.String tel
```

---

num

```
private int num
```

---

rue

```
private java.lang.String rue
```

---

code

```
private int code
```

---

ville

```
private java.lang.String ville
```

---

pays

```
private java.lang.String pays
```

Constructor Detail

Coordonnees

```
public Coordonnees(java.lang.String tel,
                   int num,
                   java.lang.String rue,
                   int code,
                   java.lang.String ville,
                   java.lang.String pays)
```

constructeur

Parameters:

`tel` - numero de telephone
`num` - numero de maison
`rue` - nom de la rue
`code` - code postal
`ville` - ville
`pays` - pays

Since:

1.2

Method Detail

getTel

```
public final java.lang.String getTel()
```

accesseur telephone

Returns:

telephone

---

getNum

```
public final int getNum()
```

accesseur numero de rue

Returns:

numero de rue

---

getRue

```
public final java.lang.String getRue()
```

accesseur nom de rue

Returns:
nom de rue

---

getCode

```
public final int getCode()
```

accesseur code postal

Returns:
code postal

---

getVille

```
public final java.lang.String getVille()
```

accesseur ville

Returns:
ville

---

getPays

```
public final java.lang.String getPays()
```

accesseur pays

Returns:
pays

Class GestionConsultation

```
java.lang.Object
  └GestionConsultation
```

```
public class GestionConsultation
extends java.lang.Object
```

classe de gestion des consultations de la clinique

Constructor Summary

GestionConsultation()

Method Summary

void	**addConsult**(java.sql.Connection conn, int NSS, java.lang.String date, float cout, int NSR, java.lang.String datefact) methode permettant d'ajouter une consultation et donc une fiche de soin
void	**addIntervConsult**(java.sql.Connection conn, int NSR, java.lang.String date, int NSS, java.lang.String nom, java.lang.String obs) methode d'ajout d'une intervention dans une consultation cette fonction met a jour les montant des fiches de soin et factures concernees ainsi que l'experience du medecin
void	**getIntervConsult**(java.sql.Connection conn, int NSS, java.lang.String date) methode permettant de lister les interventions pour une consultation donnee
void	**getListeInterv**(java.sql.Connection conn) methode permettant de lister les interventions cataloguees
int	**NSSestDsTable**(java.sql.Connection conn, int num, java.lang.String table, java.lang.String champ) methode qui permet de verifier si la donnee que l'on cherche est dans la base
(package private) static void	**printCons**(java.sql.ResultSet res, int choix) methode d'affichage des infos des consultations et des interventions
void	**setInterv**(java.sql.Connection conn, java.lang.String nom, java.lang.String matos, float cout, float remb, int dur) methode d'ajout et de modification d'une intervention dans le catalogue des interventions

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

GestionConsultation

```
public GestionConsultation()
```

Method Detail

NSSestDsTable

```
public int NSSestDsTable(java.sql.Connection conn,
                         int num,
                         java.lang.String table,
                         java.lang.String champ)
                  throws java.sql.SQLException
```

methode qui permet de verifier si la donnee que l'on cherche est dans la base

Parameters:

`conn` - le nom de la connection courante
`num` - le numero
`table` - la table ou chercher
`champ` - le champ de la table ou chercher

Returns:

1 si le numero est ds la table, 0 sinon

Throws:

`java.sql.SQLException`

---

printCons

```
static void printCons(java.sql.ResultSet res,
                      int choix)
               throws java.sql.SQLException
```

methode d'affichage des infos des consultations et des interventions

Parameters:

`res` - les n-uplets resultant de la requete
`choix` - le choix des informations que l'on veut afficher

Throws:
`java.sql.SQLException`

---

addConsult

```
public void addConsult(java.sql.Connection conn,
                       int NSS,
                       java.lang.String date,
                       float cout,
                       int NSR,
                       java.lang.String datefact)
                throws java.sql.SQLException
```

methode permettant d'ajouter une consultation et donc une fiche de soin

Parameters:
`conn` - le nom de la Connection courante
`NSS` - le numero de Securite Sociale patient
`date` - la date de la consultation
`cout` - le cout de la consultation
`NSR` - le numero d'embauche du medecin
`datefact` - la date butoire de la facture
Throws:
`java.sql.SQLException`

---

getIntervConsult

```
public void getIntervConsult(java.sql.Connection conn,
                             int NSS,
                             java.lang.String date)
                      throws java.sql.SQLException
```

methode permettant de lister les interventions pour une consultation donnee

Parameters:
`conn` - nom de la Connection courante
`NSS` - le numero de securite sociale du patient
`date` - la date de la consultation
Throws:
`java.sql.SQLException`

---

setInterv

```
public void setInterv(java.sql.Connection conn,
                      java.lang.String nom,
                      java.lang.String matos,
                      float cout,
```

```
                    float remb,
                    int dur)
             throws java.sql.SQLException
```

methode d'ajout et de modification d'une intervention dans le catalogue des interventions

Parameters:

`conn` - le nom de la connection courante
`nom` - le nom de l'intervention sur laquelle les parametres sont a modifies
`matos` - les materiaux pour l'intervention
`cout` - le cout de l'intervention
`remb` - le montant rembourse pour cette intervention
`dur` - la duree de l'intervention

Throws:

`java.sql.SQLException`

---

getListeInterv

```
public void getListeInterv(java.sql.Connection conn)
                    throws java.sql.SQLException
```

methode permettant de lister les interventions cataloguees

Parameters:

`conn` - le nom de la connexion courante

Throws:

`java.sql.SQLException`

---

addIntervConsult

```
public void addIntervConsult(java.sql.Connection conn,
                             int NSR,
                             java.lang.String date,
                             int NSS,
                             java.lang.String nom,
                             java.lang.String obs)
                      throws java.sql.SQLException
```

methode d'ajout d'une intervention dans une consultation cette fonction met a jour les montant des fiches de soin et factures concernees ainsi que l'experience du medecin

Parameters:

`conn` - le nom de la connection courante
`NSR` - le numero d'embauche du medecin
`date` - la date de la consultation
`NSS` - le numero de securite sociale du patient ayant subi l'intervention
`nom` - le nom de l'intervention a ajouter
`obs` - les observations sur l'intervention

Throws:

`java.sql.SQLException`

Class GestionPaiement

```
java.lang.Object
  └GestionPaiement
```

```
public class GestionPaiement
extends java.lang.Object
```

classe de gestion des paiements de la clinique

Constructor Summary

GestionPaiement()

Method Summary

void	getListeFactPatient(java.sql.Connection conn, int numSecuSoc) methode permettant de lister les factures pour un patient donne
void	getMensDepPatient(java.sql.Connection conn, int NSS, java.lang.String date) methode affichant les mensualites impayees d'un patient donne dont la date butoir est depassee
void	getMensFact(java.sql.Connection conn, long numFacture) methode donnant le detail des mensualites d'une facture
void	getMensImpPatient(java.sql.Connection conn, int NSS) methode affichant les mensualites impayees d'un patient donne
void	getPaiemtPatient(java.sql.Connection conn, int NSS) methode affichant la liste des paiements effectues par un patient
void	getRapPatient(java.sql.Connection conn, int NSS) methode affichant les rappels d'un patient donne dont la date butoir est depassee
int	NSSestDsTable(java.sql.Connection conn, int num, java.lang.String table, java.lang.String champ) methode qui permet de verifier si la donnee que l'on cherche est dans la base
(package private) static void	printPaiement(java.sql.ResultSet res, int choix, java.lang.String date) methode d'affichage des infos des paiements
void	setMensFact(java.sql.Connection conn, long numfact,

	java.util.LinkedList lmens) methode definissant les modalites de paiement d'une facture donnee cette fonction verifie que la somme des mensualites est bien egale au montant de la facture en question et genere les dates butoir des mensualites en fonction de celle de la facture
void	**setPaiemtPatient**(java.sql.Connection conn, int NSS, java.util.LinkedList l, java.lang.String date) methode enregistrant un paiement
void	**setRappelPatient**(java.sql.Connection conn, java.lang.String date) methode generant automatiquement les patients a partir d'une "date d'aujourd'hui" donnee (et non pas recuperee sur le systeme pour faciliter les tests) garantit qu'il n'y a qu'un rappel de type donne pour une mensualite met a jour le montant de la mensualite en fonction du type de rappel genere

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

GestionPaiement

```
public GestionPaiement()
```

Method Detail

NSSestDsTable

```
public int NSSestDsTable(java.sql.Connection conn,
                         int num,
                         java.lang.String table,
                         java.lang.String champ)
                  throws java.sql.SQLException
```

methode qui permet de verifier si la donnee que l'on cherche est dans la base

Parameters:

`conn` - le nom de la connection courante
`num` - le numero
`table` - la table ou chercher
`champ` - le champ de la table ou chercher

Returns:

1 si le numero est ds la table, 0 sinon

Throws:
`java.sql.SQLException`

---

printPaiement

```
static void printPaiement(java.sql.ResultSet res,
                          int choix,
                          java.lang.String date)
                   throws java.sql.SQLException
```

methode d'affichage des infos des paiements

Parameters:
`res` - les n-uplets resultant de la requete
`choix` - le choix des informations que l'on veut afficher
`date` - date servant a une eventuelle comparaison
Throws:
`java.sql.SQLException`

---

getListeFactPatient

```
public void getListeFactPatient(java.sql.Connection conn,
                                int numSecuSoc)
                         throws java.sql.SQLException
```

methode permettant de lister les factures pour un patient donne

Parameters:
`conn` - le nom de la connection courante
`numSecuSoc` - le numero de securite sociale du patient
Throws:
`java.sql.SQLException`

---

setMensFact

```
public void setMensFact(java.sql.Connection conn,
                        long numfact,
                        java.util.LinkedList lmens)
                 throws java.sql.SQLException
```

methode definissant les modalites de paiement d'une facture donnee
cette fonction verifie que la somme des mensualites est bien egale au montant de la facture en question
et genere les dates butoir des mensualites en fonction de celle de la facture

Parameters:

`conn` - le nom de la connection courante
`numfact` - le numero de la facture
`lmens` - detail des mensualites

Throws:
`java.sql.SQLException`

---

getMensFact

```
public void getMensFact(java.sql.Connection conn,
                        long numFacture)
                 throws java.sql.SQLException
```

methode donnant le detail des mensualites d'une facture

Parameters:
`conn` - le nom de la Connection courante
`numFacture` - le numero de la facture

Throws:
`java.sql.SQLException`

---

getMensImpPatient

```
public void getMensImpPatient(java.sql.Connection conn,
                              int NSS)
                       throws java.sql.SQLException
```

methode affichant les mensualites impayees d'un patient donne

Parameters:
`conn` - nom de la Connection courante
`NSS` - numero du patient

Throws:
`java.sql.SQLException`

---

getMensDepPatient

```
public void getMensDepPatient(java.sql.Connection conn,
                              int NSS,
                              java.lang.String date)
                       throws java.sql.SQLException
```

methode affichant les mensualites impayees d'un patient donne dont la date butoir est depassee

Parameters:
`conn` - nom de la Connection courante
`NSS` - numero du patient

`date` - mois courant "MM/YYYY"

Throws:

`java.sql.SQLException`

---

setRappelPatient

```
public void setRappelPatient(java.sql.Connection conn,
                             java.lang.String date)
                      throws java.sql.SQLException
```

methode generant automatiquement les patients a partir d'une "date d'aujourd'hui" donnee (et non pas recuperee sur le systeme pour faciliter les tests)
garantit qu'il n'y a qu'un rappel de type donne pour une mensualite
met a jour le montant de la mensualite en fonction du type de rappel genere

Parameters:

`conn` - nom de la Connection courante

`date` - "date d'aujourd'hui"

Throws:

`java.sql.SQLException`

---

getRapPatient

```
public void getRapPatient(java.sql.Connection conn,
                          int NSS)
                   throws java.sql.SQLException
```

methode affichant les rappels d'un patient donne dont la date butoir est depassee

Parameters:

`conn` - nom de la Connection courante

`NSS` - numero du patient

Throws:

`java.sql.SQLException`

---

setPaiemtPatient

```
public void setPaiemtPatient(java.sql.Connection conn,
                             int NSS,
                             java.util.LinkedList l,
                             java.lang.String date)
                      throws java.sql.SQLException
```

methode enregistrant un paiement

Parameters:

`conn` - le nom de la connection courante
`NSS` - le numero du patient
`l` - detail des mensualites visees
`date` - du paiement

Throws:
`java.sql.SQLException`

---

getPaiemtPatient

```
public void getPaiemtPatient(java.sql.Connection conn,
                             int NSS)
                      throws java.sql.SQLException
```

methode affichant la liste des paiements effectues par un patient

Parameters:
`conn` - nom de la Connection courante
`NSS` - numero du patient

Throws:
`java.sql.SQLException`

Package Class **Tree** **Deprecated** **Index** **Help**

PREV CLASS NEXT CLASS FRAMES NO FRAMES All Classes
SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

Class GestionPatient

```
java.lang.Object
  └GestionPatient
```

```
public class GestionPatient
extends java.lang.Object
```

classe de gestion des patients de la clinique

Constructor Summary

GestionPatient()

Method Summary

void	**addAllergie**(java.sql.Connection conn, java.lang.String allergie) methode servant a ajouter une allergie dans la base de donnees
void	**addAllergiePatient**(java.sql.Connection conn, int numSecuSoc, java.lang.String allergie) methode servant a la mise a jour des allergies d'un patient
void	**addCI**(java.sql.Connection conn, java.lang.String contrIndic) methode servant a ajouter une contre-indication
void	**addCIPatient**(java.sql.Connection conn, int numSecuSoc, java.lang.String contrIndic) methode servant a ajouter une contre-indication a un patient
void	**addHered**(java.sql.Connection conn, java.lang.String malHered) methode servant a ajouter une maladie hereditaire
void	**addHeredPatient**(java.sql.Connection conn, int numSecuSoc, java.lang.String malHered) methode servant a mettre a jour les maladies hereditaires d'un patient
void	**addPatient**(java.sql.Connection conn, int numSecuSoc, java.lang.String nomPatient, java.lang.String prenomPatient, java.lang.String sexePat, java.lang.String naissPat, Coordonnees coord) methode d'ajout d'un patient dans la base de donnees
void	**addProth**(java.sql.Connection conn, java.lang.String prot) methode servant a ajouter une prothese

void	**addProthPatient**(java.sql.Connection conn, int numSecuSoc, java.lang.String prot) methode servant a mettre a jour les protheses d'un patient
void	**getInfoPatient**(java.sql.Connection conn, int numSecuSoc) methode de consultation des informations administratives d'un patient
void	**getNSSPatient**(java.sql.Connection conn, java.lang.String nomPatient) methode permettant de recuperer le numero de securite sociale a partir d'un nom de patient
void	**getProfilPatient**(java.sql.Connection conn, int numSecuSoc) methode permettant l'acces au profil medical d'un patient
void	**listePatient**(java.sql.Connection conn) methode pour lister les patients soignes
int	**NomEstDsTable**(java.sql.Connection conn, java.lang.String nom, java.lang.String table, java.lang.String champ) methode qui permet de verifier si la donnee que l'on cherche est dans la base
int	**NSSestDsTable**(java.sql.Connection conn, int num, java.lang.String table, java.lang.String champ) methode qui permet de verifier si la donnee que l'on cherche est dans la base
int	**NSSetChaineEstDsTable**(java.sql.Connection conn, int num, java.lang.String type, java.lang.String table, int champ, java.lang.String champ2) methode qui permet de verifier si la donnee que l'on cherche est dans la base
(package private) static void	**printAjoutPb**(java.sql.ResultSet res) methode permettant d'afficher un probleme de sante
(package private) static void	**printPatient**(java.sql.ResultSet res, int choix) methode d'affichage des infos d'un patient
(package private) static void	**printPbSante**(java.sql.ResultSet res) methode permettant d'afficher un probleme de sante pour un patient donne
void	**setCoordPatient**(java.sql.Connection conn, int numSecuSoc, Coordonnees coord) methode de mise a jour des coordonnees d'un patient

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

GestionPatient

```
public GestionPatient()
```

Method Detail

NSSestDsTable

```
public int NSSestDsTable(java.sql.Connection conn,
                         int num,
                         java.lang.String table,
                         java.lang.String champ)
                  throws java.sql.SQLException
```

methode qui permet de verifier si la donnee que l'on cherche est dans la base

Parameters:

`conn` - le nom de la connection courante
`num` - le numero
`table` - la table ou chercher
`champ` - le champ de la table ou chercher

Returns:

1 si le numero est ds la table, 0 sinon

Throws:

`java.sql.SQLException`

NomEstDsTable

```
public int NomEstDsTable(java.sql.Connection conn,
                         java.lang.String nom,
                         java.lang.String table,
                         java.lang.String champ)
                  throws java.sql.SQLException
```

methode qui permet de verifier si la donnee que l'on cherche est dans la base

Parameters:

`conn` - le nom de la connection courante
`nom` - le nom
`table` - la table ou chercher
`champ` - le champ de la table ou chercher

Returns:

1 si le numero est ds la table, 0 sinon

Throws:

`java.sql.SQLException`

NSSetChaineEstDsTable

```
public int NSSetChaineEstDsTable(java.sql.Connection conn,
                                 int num,
                                 java.lang.String type,
                                 java.lang.String table,
                                 int champ,
                                 java.lang.String champ2)
                          throws java.sql.SQLException
```

methode qui permet de verifier si la donnee que l'on cherche est dans la base

Parameters:

`conn` - le nom de la connection courante
`num` - le numero
`type` -
`table` - la table ou chercher
`champ` - le champ de la table ou chercher
`champ2` - le deuxieme champ ou chercher

Returns:

1 si le resultat est ds la table, 0 sinon

Throws:

`java.sql.SQLException`

---

printPatient

```
static void printPatient(java.sql.ResultSet res,
                         int choix)
                  throws java.sql.SQLException
```

methode d'affichage des infos d'un patient

Parameters:

`res` - les n-uplets resultant de la requete
`choix` - le choix des attributs a afficher

Throws:

`java.sql.SQLException`

---

printAjoutPb

```
static void printAjoutPb(java.sql.ResultSet res)
                  throws java.sql.SQLException
```

methode permettant d'afficher un probleme de sante

Parameters:

`res` - les n-uplets resultat de la requete

Throws:

`java.sql.SQLException`

---

printPbSante

```
static void printPbSante(java.sql.ResultSet res)
                  throws java.sql.SQLException
```

methode permettant d'afficher un probleme de sante pour un patient donne

Parameters:
`res` - les n-uplets resultat de la requete
Throws:
`java.sql.SQLException`

---

listePatient

```
public void listePatient(java.sql.Connection conn)
                  throws java.sql.SQLException
```

methode pour lister les patients soignes

Parameters:
`conn` - le nom de la Connection courante
Throws:
`java.sql.SQLException`

---

addPatient

```
public void addPatient(java.sql.Connection conn,
                       int numSecuSoc,
                       java.lang.String nomPatient,
                       java.lang.String prenomPatient,
                       java.lang.String sexePat,
                       java.lang.String naissPat,
                       Coordonnees coord)
                throws java.sql.SQLException
```

methode d'ajout d'un patient dans la base de donnees

Parameters:
`conn` - nom de la Connection courante
`numSecuSoc` - le numero de Securite Sociale du patient
`nomPatient` - son nom
`prenomPatient` - son prenom
`sexePat` - son sexe,
`naissPat` - sa date de naissance
`coord` - ses coordonnees
Throws:
`java.sql.SQLException`

---

setCoordPatient

```
public void setCoordPatient(java.sql.Connection conn,
                            int numSecuSoc,
                            Coordonnees coord)
                     throws java.sql.SQLException
```

methode de mise a jour des coordonnees d'un patient

Parameters:

`conn` - nom de la Connection courante
`numSecuSoc` - le numero de Securite Sociale du patient
`coord` - ses nouvelles coordonnees

Throws:

`java.sql.SQLException`

---

getNSSPatient

```
public void getNSSPatient(java.sql.Connection conn,
                          java.lang.String nomPatient)
                   throws java.sql.SQLException
```

methode permettant de recuperer le numero de securite sociale a partir d'un nom de patient

Parameters:

`conn` - nom de la Connection courante
`nomPatient` - le nom du patient

Throws:

`java.sql.SQLException`

---

getInfoPatient

```
public void getInfoPatient(java.sql.Connection conn,
                           int numSecuSoc)
                    throws java.sql.SQLException
```

methode de consultation des informations administratives d'un patient

Parameters:

`conn` - le nom de la Connection courante
`numSecuSoc` - le numero de Securite Sociale du patient

Throws:

`java.sql.SQLException`

---

addAllergie

```
public void addAllergie(java.sql.Connection conn,
                        java.lang.String allergie)
                 throws java.sql.SQLException
```

methode servant a ajouter une allergie dans la base de donnees

Parameters:

`conn` - le nom de la Connection courante

`allergie` - le nom de cette allergie

Throws:

`java.sql.SQLException`

---

addCI

```
public void addCI(java.sql.Connection conn,
                  java.lang.String contrIndic)
           throws java.sql.SQLException
```

methode servant a ajouter une contre-indication

Parameters:

`conn` - le nom de la connection courante

`contrIndic` - le nom de cette contre-indication

Throws:

`java.sql.SQLException`

---

addProth

```
public void addProth(java.sql.Connection conn,
                     java.lang.String prot)
              throws java.sql.SQLException
```

methode servant a ajouter une prothese

Parameters:

`conn` - le nom de la Connection courante

`prot` - le nom de la prothese

Throws:

`java.sql.SQLException`

---

addHered

```
public void addHered(java.sql.Connection conn,
                     java.lang.String malHered)
              throws java.sql.SQLException
```

methode servant a ajouter une maladie hereditaire

Parameters:

`conn` - le nom de la Connection courante

`malHered` - le nom de cette maladie

Throws:

`java.sql.SQLException`

---

addAllergiePatient

```
public void addAllergiePatient(java.sql.Connection conn,
                               int numSecuSoc,
                               java.lang.String allergie)
                        throws java.sql.SQLException
```

methode servant a la mise a jour des allergies d'un patient

Parameters:

`conn` - le nom de la Connection courante

`numSecuSoc` - le numero de securite sociale du patient

`allergie` - le nom de cette allergie

Throws:

`java.sql.SQLException`

---

addCIPatient

```
public void addCIPatient(java.sql.Connection conn,
                         int numSecuSoc,
                         java.lang.String contrIndic)
                  throws java.sql.SQLException
```

methode servant a ajouter une contre-indication a un patient

Parameters:

`conn` - le nom de la Connection courante

`numSecuSoc` - le numero de securite sociale du patient

`contrIndic` - le nom de cette contre-indication

Throws:

`java.sql.SQLException`

---

addProthPatient

```
public void addProthPatient(java.sql.Connection conn,
                            int numSecuSoc,
                            java.lang.String prot)
                     throws java.sql.SQLException
```

methode servant a mettre a jour les protheses d'un patient

Parameters:

`conn` - le nom de la Connection courante
`numSecuSoc` - le numero de securite sociale
`prot` - le nom de la prothese

Throws:

`java.sql.SQLException`

---

addHeredPatient

```
public void addHeredPatient(java.sql.Connection conn,
                            int numSecuSoc,
                            java.lang.String malHered)
                     throws java.sql.SQLException
```

methode servant a mettre a jour les maladies hereditaires d'un patient

Parameters:

`conn` - nom de la Connection courante
`numSecuSoc` - le numero de securite sociale
`malHered` - le nom de cette maladie

Throws:

`java.sql.SQLException`

---

getProfilPatient

```
public void getProfilPatient(java.sql.Connection conn,
                             int numSecuSoc)
                      throws java.sql.SQLException
```

methode permettant l'acces au profil medical d'un patient

Parameters:

`conn` - le nom de la connection courante
`numSecuSoc` - le numero de securite sociale du patient

Throws:

`java.sql.SQLException`

Class GestionPersonnel

```
java.lang.Object
  └GestionPersonnel
```

```
public class GestionPersonnel
extends java.lang.Object
```

classe de gestion du personnel de la clinique

Constructor Summary

GestionPersonnel()

Method Summary

void	**addAdmin**(java.sql.Connection conn, int numEmbauche, java.lang.String nomPersAdmin, java.lang.String prenomPersAdmin, java.lang.String adminJob, float salAdmin, Coordonnees coord) methode permettant d'ajouter un membre du personnel administratif
void	**addAux**(java.sql.Connection conn, int numEmbauche, java.lang.String nomPersAux, java.lang.String prenomPersAux, java.lang.String auxJob, float tauxHAux, Coordonnees coord) methode permettant d'ajouter un auxiliaire medical
void	**addExpMed**(java.sql.Connection conn, int NSR, java.lang.String interv) methode permettant d'ajouter une intervention a l'experience d'un medecin Nous distinguons le cas d'un gain d'experience pour une intervention donnee et le cas d'une nouvelle intervention encore jamais realisee par ce medecin
void	**addMed**(java.sql.Connection conn, int numEmbauche, java.lang.String nomPersMed, java.lang.String prenomPersMed, java.lang.String medJob, float tauxHMed, Coordonnees coord) methode permettant d'ajouter un medecin
void	**calcSalPers**(java.sql.Connection conn, int NSR, java.lang.String mois) methode calculant le salaire d'un mois donne d'un membre du personnel
void	**getExpMed**(java.sql.Connection conn, int NSR) methode permettant d'acceder a l'experience d'un medecin
void	**getInfoPers**(java.sql.Connection conn, int numEmbauche)

	methode permettant d'acceder aux informations d'un membre du personnel a partir de son NSR
void	**getNSRPers**(java.sql.Connection conn, java.lang.String persNom) methode permettant d'acceder au numero d'embauche d'un membre du personnel a partir de son nom
void	**listePers**(java.sql.Connection conn, int choix) methode permettant de lister le personnel de la clinique
int	**NomEstDsTable**(java.sql.Connection conn, java.lang.String nom, java.lang.String table, java.lang.String champ) methode qui permet de verifier si la donnee que l'on cherche est dans la base
int	**NSRestDsTable**(java.sql.Connection conn, int num, java.lang.String table, java.lang.String champ) methode qui permet de verifier si la donnee que l'on cherche est dans la base
(package private) static void	**printPers**(java.sql.ResultSet res, int choix) methode d'affichage des infos du personnel
void	**rmPers**(java.sql.Connection conn, int numEmbauche, int choix) methode permettant de supprimer un membre du personnel
void	**setCoordPers**(java.sql.Connection conn, int numEmbauche, Coordonnees coord) methode de mise a jour des coordonnees d'un membre du personnel
void	**setProfPers**(java.sql.Connection conn, int NSR, java.lang.String prof) methode permettant la modification de la profession d'un membre du personnel
void	**setSalPers**(java.sql.Connection conn, int numEmbauche, int choix, float sal) methode permettant la modification du salaire ou du taux horaire d'un membre du personnel

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

GestionPersonnel

public **GestionPersonnel**()

Method Detail

NSRestDsTable

```
public int NSRestDsTable(java.sql.Connection conn,
                         int num,
                         java.lang.String table,
                         java.lang.String champ)
                  throws java.sql.SQLException
```

methode qui permet de verifier si la donnee que l'on cherche est dans la base

Parameters:

`conn` - le nom de la connection courante
`num` - le numero
`table` - la table ou chercher
`champ` - le champ de la table ou chercher

Returns:

1 si le numero est ds la table, 0 sinon

Throws:

`java.sql.SQLException`

NomEstDsTable

```
public int NomEstDsTable(java.sql.Connection conn,
                         java.lang.String nom,
                         java.lang.String table,
                         java.lang.String champ)
                  throws java.sql.SQLException
```

methode qui permet de verifier si la donnee que l'on cherche est dans la base

Parameters:

`conn` - le nom de la connection courante
`nom` - le nom
`table` - la table ou chercher
`champ` - le champ de la table ou chercher

Returns:

1 si le numero est ds la table, 0 sinon

Throws:

`java.sql.SQLException`

---

printPers

```
static void printPers(java.sql.ResultSet res,
                      int choix)
               throws java.sql.SQLException
```

methode d'affichage des infos du personnel

Parameters:

`res` - les n-uplets resultant de la requete

`choix` - le choix des attributs a afficher

Throws:

`java.sql.SQLException`

---

listePers

```
public void listePers(java.sql.Connection conn,
                      int choix)
               throws java.sql.SQLException
```

methode permettant de lister le personnel de la clinique

Parameters:

`conn` - nom de la connection courante

`choix` - le choix de la categorie du personnel a lister

Throws:

`java.sql.SQLException`

---

addAdmin

```
public void addAdmin(java.sql.Connection conn,
                     int numEmbauche,
                     java.lang.String nomPersAdmin,
                     java.lang.String prenomPersAdmin,
                     java.lang.String adminJob,
                     float salAdmin,
                     Coordonnees coord)
              throws java.sql.SQLException
```

methode permettant d'ajouter un membre du personnel administratif

Parameters:

`conn` - nom de la Connection courante

`numEmbauche` - le numero d'embauche de ce membre

`nomPersAdmin` - son nom

`prenomPersAdmin` - son prenom

`adminJob` - sa profession

`salAdmin` - son salaire

`coord` - ses coordonnees

Throws:

`java.sql.SQLException`

---

addAux

```
public void addAux(java.sql.Connection conn,
                   int numEmbauche,
                   java.lang.String nomPersAux,
```

```
                java.lang.String prenomPersAux,
                java.lang.String auxJob,
                float tauxHAux,
                Coordonnees coord)
        throws java.sql.SQLException
```

methode permettant d'ajouter un auxiliaire medical

Parameters:

`conn` - nom de la connection courante
`numEmbauche` - le numero donne a l'embauche a l'auxiliaire
`nomPersAux` - son nom
`prenomPersAux` - son prenom
`auxJob` - sa profession
`tauxHAux` - son taux horaire
`coord` - ses coordonnees

Throws:

`java.sql.SQLException`

---

addMed

```
public void addMed(java.sql.Connection conn,
                int numEmbauche,
                java.lang.String nomPersMed,
                java.lang.String prenomPersMed,
                java.lang.String medJob,
                float tauxHMed,
                Coordonnees coord)
        throws java.sql.SQLException
```

methode permettant d'ajouter un medecin

Parameters:

`conn` - le nom de la Connection courante,
`numEmbauche` - le numero d'embauche du medecin,
`nomPersMed` - son nom,
`prenomPersMed` - son prenom,
`medJob` - sa profession,
`tauxHMed` - son taux horaire,
`coord` - ses coordonnees

Throws:

`java.sql.SQLException`

---

rmPers

```
public void rmPers(java.sql.Connection conn,
                int numEmbauche,
                int choix)
```

```
        throws java.sql.SQLException
```

methode permettant de supprimer un membre du personnel

Parameters:

`conn` - le nom de la Connection courante,
`numEmbauche` - le numero de securite sociale de ce membre,
`choix` - le choix de la categorie du personnel a supprimer

Throws:

`java.sql.SQLException`

---

setCoordPers

```
public void setCoordPers(java.sql.Connection conn,
                         int numEmbauche,
                         Coordonnees coord)
                  throws java.sql.SQLException
```

methode de mise a jour des coordonnees d'un membre du personnel

Parameters:

`conn` - nom de la Connection courante
`numEmbauche` - le numero d'embauche du membre
`coord` - ses coordonnees

Throws:

`java.sql.SQLException`

---

getNSRPers

```
public void getNSRPers(java.sql.Connection conn,
                       java.lang.String persNom)
                throws java.sql.SQLException
```

methode permettant d'acceder au numero d'embauche d'un membre du personnel a partir de son nom

Parameters:

`conn` - le nom de la connection courante
`persNom` - le nom du membre en question

Throws:

`java.sql.SQLException`

---

getInfoPers

```
public void getInfoPers(java.sql.Connection conn,
                        int numEmbauche)
                 throws java.sql.SQLException
```

methode permettant d'acceder aux informations d'un membre du personnel a partir de son NSR

Parameters:

`conn` - le nom de la Connection courante,
`numEmbauche` - le numero d'embauche du membre

Throws:

`java.sql.SQLException`

---

setSalPers

```
public void setSalPers(java.sql.Connection conn,
                       int numEmbauche,
                       int choix,
                       float sal)
                throws java.sql.SQLException
```

methode permettant la modification du salaire ou du taux horaire d'un membre du personnel

Parameters:

`conn` - le nom de la Connection courante
`numEmbauche` - le numero d'embauche de ce membre
`choix` - le choix de la categorie du personnel a mettre a jour
`sal` - le montant du nouveau salaire

Throws:

`java.sql.SQLException`

---

calcSalPers

```
public void calcSalPers(java.sql.Connection conn,
                        int NSR,
                        java.lang.String mois)
                 throws java.sql.SQLException
```

methode calculant le salaire d'un mois donne d'un membre du personnel

Parameters:

`conn` - le nom de la connection courante
`NSR` - le numero de securite sociale de ce membre
`mois` - le mois d'une annee donnee pour lequel on veut le salaire

Throws:

`java.sql.SQLException`

---

setProfPers

```
public void setProfPers(java.sql.Connection conn,
                        int NSR,
                        java.lang.String prof)
```

```
                throws java.sql.SQLException
```

methode permettant la modification de la profession d'un membre du personnel

Parameters:

`conn` - le nom de la Connection courante
`NSR` - le numero d'embauche de ce membre
`prof` - la nouvelle profession du membre du personnel

Throws:

`java.sql.SQLException`

---

getExpMed

```
public void getExpMed(java.sql.Connection conn,
                      int NSR)
               throws java.sql.SQLException
```

methode permettant d'acceder a l'experience d'un medecin

Parameters:

`conn` - le nom de la connexion courante
`NSR` - le numero d'embauche du medecin

Throws:

`java.sql.SQLException`

---

addExpMed

```
public void addExpMed(java.sql.Connection conn,
                      int NSR,
                      java.lang.String interv)
               throws java.sql.SQLException
```

methode permettant d'ajouter une intervention a l'experience d'un medecin Nous distinguons le cas d'un gain d'experience pour une intervention donnee et le cas d'une nouvelle intervention encore jamais realisee par ce medecin

Parameters:

`conn` - le nom de la connexion courante
`NSR` - le numero d'embauche du medecin
`interv` - le nom de l'interv

Throws:

`java.sql.SQLException`

Class Historique

```
java.lang.Object
  └Historique
```

```
public class Historique
extends java.lang.Object
```

classe permettant d'avoir l'historique des soins pour un patient

Constructor Summary

Historique()

Method Summary

void	**getListeConsult**(java.sql.Connection conn) methode permettant de lister toutes les fiches de soin
void	**getListeConsultPatient**(java.sql.Connection conn, int numSecuSoc) methode permettant de lister les consultations pour un patient donne
void	**getListeInterv**(java.sql.Connection conn) methode permettant de lister toutes les interventions
void	**getListeIntervPatient**(java.sql.Connection conn, int numSecuSoc) methode permettant de lister les interventions pour un patient donne
int	**NSSestDsTable**(java.sql.Connection conn, int num, java.lang.String table, java.lang.String champ) methode qui permet de verifier si la donnee que l'on cherche est dans la base
(package private) static void	**printHistorique**(java.sql.ResultSet res, int choix) methode d'affichage de l'historique des soins

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

Historique

```
public Historique()
```

Method Detail

NSSestDsTable

```
public int NSSestDsTable(java.sql.Connection conn,
                         int num,
                         java.lang.String table,
                         java.lang.String champ)
                  throws java.sql.SQLException
```

methode qui permet de verifier si la donnee que l'on cherche est dans la base

Parameters:

`conn` - le nom de la connection courante
`num` - le numero
`table` - la table ou chercher
`champ` - le champ de la table ou chercher

Returns:

1 si le numero est ds la table, 0 sinon

Throws:

`java.sql.SQLException`

---

printHistorique

```
static void printHistorique(java.sql.ResultSet res,
                            int choix)
                     throws java.sql.SQLException
```

methode d'affichage de l'historique des soins

Parameters:

`res` - les n-uplets resultant de la requete
`choix` - le choix des informations que l'on veut afficher (consultations, interventions, factures)

Throws:

`java.sql.SQLException`

---

getListeConsult

```
public void getListeConsult(java.sql.Connection conn)
```

```
                    throws java.sql.SQLException
```

methode permettant de lister toutes les fiches de soin

Parameters:

`conn` - le nom de la connection courante

Throws:

`java.sql.SQLException`

---

getListeConsultPatient

```
public void getListeConsultPatient(java.sql.Connection conn,
                                   int numSecuSoc)
                            throws java.sql.SQLException
```

methode permettant de lister les consultations pour un patient donne

Parameters:

`conn` - le nom de la connection courante

`numSecuSoc` - le numero de securite sociale du patient

Throws:

`java.sql.SQLException`

---

getListeInterv

```
public void getListeInterv(java.sql.Connection conn)
                    throws java.sql.SQLException
```

methode permettant de lister toutes les interventions

Parameters:

`conn` - le nom de la connection courante

Throws:

`java.sql.SQLException`

---

getListeIntervPatient

```
public void getListeIntervPatient(java.sql.Connection conn,
                                  int numSecuSoc)
                           throws java.sql.SQLException
```

methode permettant de lister les interventions pour un patient donne

Parameters:

`conn` - le nom de la connection courante

`numSecuSoc` - le numero de securite sociale du patient

Throws:

`java.sql.SQLException`

Package Class **Tree** **Deprecated** **Index** **Help**

PREV CLASS NEXT CLASS FRAMES NO FRAMES All Classes

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

Class Parametres

```
java.lang.Object
  └ Parametres
```

```
public final class Parametres
extends java.lang.Object
```

classe de recuperation et de formatage des parametres a passer dans les requetes librement inspiree de la classe EasyIn disponible sur le kiosk

Field Summary

private static java.io.BufferedReader	**br**

Constructor Summary

Parametres()

Method Summary

static java.lang.String	**add15Jours**(java.lang.String date) ajoute 15 jours a une date si le temps le premet a ameliorer en gerant les mois de 30/31/28 jrs et les annees bissextiles
static long	**dateToLong**(java.lang.String date) conversion d'une date en un entier
static java.lang.String	**dateToString**(long date) conversion d'un entier en une date
static java.lang.String	**decrMois**(java.lang.String date) decremente le mois d'une date
static int	**getAnnee**(java.lang.String prompt) lecture d'une annee
static java.lang.String	**getChaine**(java.lang.String prompt, int vide) lecture d'une chaine de caracteres (nom, prenom, nom

	d'intervention, materiaux, prothese etc.)
static int	**getCode**(java.lang.String prompt) lecture d'un code postal
static Coordonnees	**getCoord**() lecture des coordonnees d'un patient
static java.lang.String	**getDate**(java.lang.String prompt, int format) lecture d'une date
static int	**getJM**(java.lang.String prompt, int max) lecture d'un jour ou d'un mois
static java.util.LinkedList	**getMens**() lecture d'une liste de mensualites attention cette liste contient des objets, donc des Float et des Integer
static float	**getMont**(java.lang.String prompt) lecture d'un montant (de facture, de mensualite, remourse, taux horaire etc.)
static int	**getNum**(java.lang.String prompt, int nul) lecture d'un numero
static long	**getNumFact**(java.lang.String prompt, int nul) lecture d'un numero long
static java.util.LinkedList	**getPaiemt**() lecture du detail d'un paiement
static java.lang.String	**getSexe**(java.lang.String prompt) lecture d'un sexe
static java.lang.String	**getTel**(java.lang.String prompt) lecture d'un numero de telephone
private static java.util.StringTokenizer	**getToken**()
static java.lang.String	**incrMois**(java.lang.String date) incremente le mois d'une date
static void	**main**(java.lang.String[] arg)

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

br

```
private static final java.io.BufferedReader br
```

Constructor Detail

Parametres

```
public Parametres()
```

Method Detail

getToken

```
private static final java.util.StringTokenizer getToken()
                                                 throws java.io.IOException
```

Throws:
`java.io.IOException`

---

getNum

```
public static final int getNum(java.lang.String prompt,
                               int nul)
```

lecture d'un numero

Parameters:
`prompt` - le prompt
`nul` - different de 0 si l'entier peut etre nul
Returns:
le numero (positif)

---

getNumFact

```
public static final long getNumFact(java.lang.String prompt,
                                    int nul)
```

lecture d'un numero long

Parameters:
`prompt` - le prompt
`nul` - different de 0 si l'entier peut etre nul
Returns:

le numero (positif)

---

getChaine

```
public static final java.lang.String getChaine(java.lang.String prompt,
                                                int vide)
```

lecture d'une chaine de caracteres (nom, prenom, nom d'intervention, materiaux, prothese etc.)

Parameters:
`prompt` - le prompt
`vide` - different de 0 si la chaine peut etre vide
Returns:
la chaine

---

getMont

```
public static final float getMont(java.lang.String prompt)
```

lecture d'un montant (de facture, de mensualite, remourse, taux horaire etc.)

Parameters:
`prompt` - le prompt
Returns:
le montant

---

getTel

```
public static final java.lang.String getTel(java.lang.String prompt)
```

lecture d'un numero de telephone

Parameters:
`prompt` - le prompt
Returns:
le numero de telephone

---

getCode

```
public static final int getCode(java.lang.String prompt)
```

lecture d'un code postal

Parameters:

`prompt` - le prompt

Returns:

le code postal (5 chiffres max)

---

getSexe

```
public static final java.lang.String getSexe(java.lang.String prompt)
```

lecture d'un sexe

Parameters:

`prompt` - le prompt

Returns:

le sexe (M/F)

---

getCoord

```
public static final Coordonnees getCoord()
```

lecture des coordonnees d'un patient

Returns:

un objet Coordonnees

See Also:

`Coordonnees`

---

getJM

```
public static final int getJM(java.lang.String prompt,
                              int max)
```

lecture d'un jour ou d'un mois

Parameters:

`prompt` - le prompt

`max` - valeur maximale attendue (typiquement 31 ou 12)

Returns:

le nombre

---

getAnnee

```
public static final int getAnnee(java.lang.String prompt)
```

lecture d'une annee

Parameters:
`prompt` - le prompt
Returns:
l'annee

---

getDate

```
public static final java.lang.String getDate(java.lang.String prompt,
                                              int format)
```

lecture d'une date

Parameters:
`prompt` - le prompt
`format` - formmat de la date (2 : m/a sinon j/m/a)
Returns:
une chaine "j/m/a" ou "m/a"

---

getMens

```
public static final java.util.LinkedList getMens()
```

lecture d'une liste de mensualites
attention cette liste contient des objets, donc des Float et des Integer

Returns:
la liste : nb de mens (int) -> total des mens (float) -> mens1 (int) -> mens2 (int) -> ...

---

dateToLong

```
public static final long dateToLong(java.lang.String date)
```

conversion d'une date en un entier

Parameters:
`date` - chaine "DD/MM/YYYY"
Returns:
entier YYYYMMDD
Since:
2.0

---

dateToString

```
public static final java.lang.String dateToString(long date)
```

conversion d'un entier en une date

Parameters:
`date` - l'entier YYYYMMDD
Returns:
chaine "DD/MM/YYYY"
Since:
2.1

---

decrMois

```
public static final java.lang.String decrMois(java.lang.String date)
```

decremente le mois d'une date

Parameters:
`date` - chaine "DD/MM/YYYY"
Returns:
chaine "DD/MM-1/YYYY"
Since:
2.1

---

add15Jours

```
public static final java.lang.String add15Jours(java.lang.String date)
```

ajoute 15 jours a une date
si le temps le premet a ameliorer en gerant les mois de 30/31/28 jrs et les annees bissextiles

Parameters:
`date` - chaine "DD/MM/YYYY"
Returns:
chaine "DD+15/MM/YYYY"
Since:
2.2

---

incrMois

```
public static final java.lang.String incrMois(java.lang.String date)
```

incremente le mois d'une date

Parameters:

`date` - chaine "DD/MM/YYYY"

Returns:

chaine "DD/MM+1/YYYY"

Since:

2.2

---

getPaiemt

```
public static final java.util.LinkedList getPaiemt()
```

lecture du detail d'un paiement

Returns:

la liste : nb de mens visees (int) -> no_fact_mens1 (long) -> date_mens1 (string) -> ...

---

main

```
public static void main(java.lang.String[] arg)
```

Class Statistiques

```
java.lang.Object
  └Statistiques
```

```
public class Statistiques
extends java.lang.Object
```

classe permettant d'acceder a des statistiques sur la clinique

Constructor Summary

Statistiques()

Method Summary

void	**moyConsultJour**(java.sql.Connection conn) methode de calcul du nombre moyen de consultations par jour
void	**moyConsultMois**(java.sql.Connection conn) methode de calcul du nombre moyen de consultations par mois
void	**nbPatient**(java.sql.Connection conn) methode de calcul du nombre de patient dans la clinique
void	**nbPatientFactImp**(java.sql.Connection conn) methode de calcul du nombre de patient ayant des factures impayes
void	**nbPers**(java.sql.Connection conn) methode de calcul du nombre de personnel dans la clinique

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

Statistiques

```
public Statistiques()
```

Method Detail

moyConsultJour

```
public void moyConsultJour(java.sql.Connection conn)
                    throws java.sql.SQLException
```

methode de calcul du nombre moyen de consultations par jour

Parameters:
`conn` - le nom de la connection courante
Throws:
`java.sql.SQLException`

---

moyConsultMois

```
public void moyConsultMois(java.sql.Connection conn)
                    throws java.sql.SQLException
```

methode de calcul du nombre moyen de consultations par mois

Parameters:
`conn` - le nom de la connection courante
Throws:
`java.sql.SQLException`

---

nbPatientFactImp

```
public void nbPatientFactImp(java.sql.Connection conn)
                      throws java.sql.SQLException
```

methode de calcul du nombre de patient ayant des factures impayes

Parameters:
`conn` - le nom de la connection courante
Throws:
`java.sql.SQLException`

---

nbPatient

```
public void nbPatient(java.sql.Connection conn)
               throws java.sql.SQLException
```

methode de calcul du nombre de patient dans la clinique

Parameters:

`conn` - le nom de la connection courante

Throws:

`java.sql.SQLException`

---

nbPers

```
public void nbPers(java.sql.Connection conn)
            throws java.sql.SQLException
```

methode de calcul du nombre de personnel dans la clinique

Parameters:

`conn` - le nom de la connection courante

Throws:

`java.sql.SQLException`
