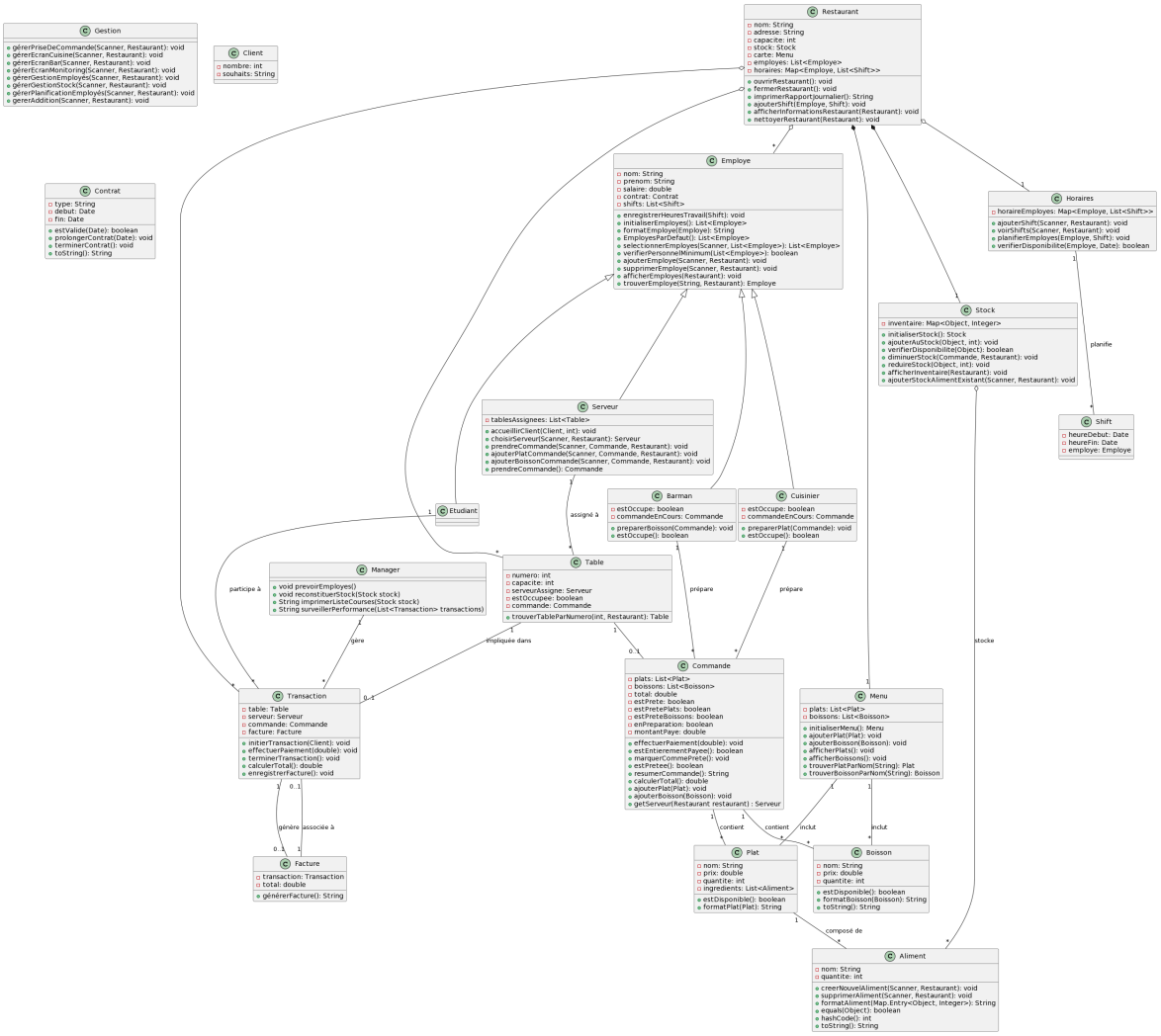
PROJET JAVA - SYSTÈME DE GESTION DE RESTAURANT

DOCUMENTATION :

- **Introduction**
  - Ce projet Java est une application de gestion de restaurant conçue pour automatiser et faciliter les opérations quotidiennes d'un établissement de restauration.
    Elle couvre des aspects tels que la prise de commandes, la gestion des stocks, le suivi des employés et des transactions financières.
- **Prérequis**
  - Java Development Kit (JDK) version 11 ou ultérieure.
    Un environnement de développement intégré (IDE) comme Eclipse, IntelliJ IDEA, ou équivalent.
- **Installation**
  - Téléchargement du projet : Téléchargez le dossier du projet.
    Importation dans Eclipse :
    Ouvrez Eclipse.
    Sélectionnez 'File' > 'Import'.
    Choisissez 'Existing Projects into Workspace'.
    Naviguez jusqu'au dossier téléchargé et importez-le.
- **Guide d'Utilisation**
  - Lancement de l'Application
    Dans Eclipse, naviguez jusqu'au dossier 'src'.
    Trouvez le fichier App.java.
    Faites un clic droit sur App.java et sélectionnez 'Run As' > 'Java Application'.
- Navigation dans l'Application
  - L'application démarre avec un menu principal offrant différentes options liées à la gestion du restaurant. Utilisez le clavier pour naviguer dans le menu et sélectionner les fonctions.

DIAGRAMME DE CLASSES



Code UML : “Diagramme de classes UML - Code.txt”

Diagramme UML : “Diagramme de classes UML.png”

DIFFÉRENTES ENTITÉS ET LEURS RELATIONS

CLASSE RESTAURANT

- Attributs:
  - **nom**: Le nom du restaurant.
  - **adresse**: L'adresse physique du restaurant.
  - **capacite**: La capacité d'accueil du restaurant.
  - **stock**: Référence à un objet **Stock** qui gère l'inventaire des aliments et boissons.

    - **carte**: Référence à un objet **Menu** représentant les plats et boissons proposés.
    - **employes**: Liste des employés (serveurs, cuisiniers, etc.) du restaurant.
    - **horaires**: Une carte associant chaque employé à ses horaires de travail (**Shift**).
- Méthodes:
    - **ouvrirRestaurant()**: Prépare le restaurant pour l'ouverture, incluant la gestion des stocks et la planification des employés.
    - **fermerRestaurant()**: Gère les opérations de fermeture du restaurant.
    - **imprimerRapportJournalier()**: Génère un rapport des activités et ventes journalières.
    - **getHoraires()**: Retourne la planification des horaires de tous les employés.
    - **afficherInformationsRestaurant(Restaurant)**: Affiche des informations détaillées sur le restaurant.
    - **nettoyerRestaurant(Restaurant)**: Gère le processus de nettoyage du restaurant après fermeture.
    - ajouterShift(Employe employe, Shift shift): méthode pour remplir le planning des horaires par le shift de l'employé.
    - Les méthodes **get** et **set** pour les attributs.

CLASSE EMPLOYE

- Attributs:
    - **nom**, **prenom**: Nom et prénom de l'employé.
    - **salaire**: Salaire de l'employé.
    - **contrat**: Référence à un objet **Contrat** détaillant les termes du contrat de l'employé.
    - **shifts**: Liste des shifts (horaires de travail) de l'employé.
- Méthodes:
    - **enregistrerHeuresTravail(Shift)**: Enregistre les heures de travail d'un employé.
    - **getShifts()**, **setShifts(List<Shift>)**: Gestion des shifts de l'employé.
    - **initialiserEmployes()**: Initialise une liste d'employés par défaut.
    - **formatEmploye(Employe)**: Formate l'information d'un employé pour l'affichage.
    - **EmployesParDefaut()**: Retourne une liste prédéfinie d'employés.
    - **selectionnerEmployes(Scanner, List<Employe>)**: Permet de sélectionner des employés pour une journée de travail.

- verifierPersonnelMinimum(List<Employe>): Vérifie si le nombre minimum d'employés requis est présent.
- ajouterEmploye(Scanner, Restaurant), supprimerEmploye(Scanner, Restaurant), afficherEmployes(Restaurant): Gestion des employés
- trouverEmploye(String, Restaurant): Trouve un employé par son nom.
- Les méthodes get et set pour les autres attributs.

CLASSE SERVEUR (ÉTEND EMPLOYE)

• Attributs:
  - tablesAssignees: Liste des tables assignées au serveur.
• Méthodes:
  - accueillirClient(Client, int): Gère l'accueil des clients.
  - choisirServeur(Scanner, Restaurant): Permet de choisir un serveur pour une table.
  - prendreCommande(Scanner, Commande, Restaurant): Gère la prise de commande des clients.
  - ajouterPlatCommande(Scanner, Commande, Restaurant), ajouterBoissonCommande(Scanner, Commande, Restaurant): Ajoute un plat ou une boisson à une commande.
  - getTablesAssignees(), setTablesAssignees(List<Table>): Gestion des tables assignées.
  - prendreCommande(): Crée une nouvelle commande.

CLASSE CUISINIER (ÉTEND EMPLOYE)

• Attributs:
  - estOccupe: Indique si le cuisinier est actuellement occupé à préparer une commande.
  - commandeEnCours: La commande que le cuisinier est en train de préparer.
• Méthodes:
  - preparerPlat(Commande): Prépare les plats d'une commande.
  - estOccupe(), setOccupe(boolean): Gestion de l'état occupé du cuisinier.
  - getCommandeEnCours(), setCommandeEnCours(Commande): Gestion de la commande en cours de préparation.
  - estOccupeAvecCommande(Commande): Vérifie si le cuisinier est occupé avec une commande spécifique.

CLASSE BARMAN (ÉTEND EMPLOYE)

- **Attributs** et **Méthodes** similaires à **Cuisinier**, mais pour la préparation de boissons.

CLASSE MANAGER (ÉTEND EMPLOYE)

- **Méthodes spécifiques**:
  - **prevoirEmployes()**: Prévoit les employés nécessaires pour une journée.
  - **reconstituerStock(Stock)**: Gère la reconstitution du stock.
  - **imprimerListeCourses(Stock)**: Imprime une liste de courses basée sur le stock.
  - **surveillerPerformance(List<Transaction>)**: Surveille et évalue les performances du restaurant.

CLASSE ETUDIANT (ÉTEND EMPLOYE)

- Pas d'attributs ou de méthodes supplémentaires, représente un employé étudiant avec des conditions contractuelles potentiellement différentes.

CLASSE GESTION

- Contient des méthodes statiques pour la gestion de différentes opérations dans le restaurant (prise de commande, gestion de cuisine, bar, monitoring, etc).

CLASSE STOCK

- **Attributs**:
  - **inventaire**: Map stockant les objets (Plats, Boissons, Aliments) et leurs quantités.
- **Méthodes**:
  - **initialiserStock()**: Initialise le stock avec des aliments par défaut.
  - **ajouterAuStock(Object, int)**: Ajoute ou met à jour un item dans le stock.
  - **verifierDisponibilite(Object)**: Vérifie si un item est disponible en stock.

- **diminuerStock(Commande, Restaurant)**: Diminue le stock en fonction des commandes.
- **reduireStock(Object, int)**: Réduit la quantité d'un item dans le stock.
- **afficherInventaire(Restaurant)**: Affiche l'inventaire actuel du restaurant.
- **ajouterStockAlimentExistant(Scanner, Restaurant)**: Ajoute du stock pour un aliment existant.
- **getInventaire()**, **setInventaire(Map<Object, Integer>)**: Gestion de l'inventaire.

CLASSE MENU

- Attributs:
  - **plats**: Liste des plats disponibles dans le menu.
  - **boissons**: Liste des boissons disponibles.
- Méthodes:
  - **initialiserMenu()**: Initialise le menu avec des plats et boissons.
  - **ajouterPlat(Plat)**, **ajouterBoisson(Boisson)**: Ajoute un plat ou une boisson au menu.
  - **getPlats()**, **setPlats(List<Plat>)**: Gestion des plats.
  - **getBoissons()**, **setBoissons(List<Boisson>)**: Gestion des boissons.
  - **afficherPlats()**, **afficherBoissons()**: Affiche les plats et boissons disponibles.
  - **trouverPlatParNom(String)**, **trouverBoissonParNom(String)**: Trouve un plat ou une boisson par son nom.

CLASSE PLAT

- Attributs:
  - **nom**: Nom du plat.
  - **prix**: Prix du plat.
  - **quantite**: Quantité disponible du plat.
  - **ingredients**: Liste des ingrédients composant le plat.
- Méthodes:
  - **estDisponible()**: Vérifie si le plat est disponible.
  - **formatPlat(Plat)**: Formate l'information d'un plat.
  - **toString()**: Représentation en chaîne du plat.
  - Les méthodes **get** et **set** pour les attributs.

CLASSE BOISSON

- Attributs:
  - nom: Le nom de la boisson.
  - prix: Le prix de la boisson.
  - quantite: La quantité disponible de la boisson.
- Méthodes:
  - estDisponible(): Retourne un booléen indiquant si la boisson est disponible en stock.
  - formatBoisson(Boisson): Formate les informations de la boisson pour l'affichage.
  - toString(): Renvoie une représentation sous forme de chaîne de la boisson.
  - Les méthodes get et set pour les attributs.

CLASSE ALIMENT

- Attributs:
  - nom: Le nom de l'aliment.
  - quantite: La quantité disponible de l'aliment.
- Méthodes:
  - creerNouvelAliment(Scanner, Restaurant): Permet de créer un nouvel aliment et de l'ajouter au stock du restaurant.
  - supprimerAliment(Scanner, Restaurant): Permet de supprimer un aliment du stock.
  - formatAliment(Map.Entry<Object, Integer>): Formate les informations de l'aliment pour l'affichage.
  - Les méthodes get et set pour les attributs.
  - equals(Object), hashCode(): Méthodes pour comparer et hasher les objets Aliment.

CLASSE CLIENT

- Attributs:
  - nombre: Le nombre de clients dans le groupe.
  - souhaits: Les souhaits spécifiques exprimés par le groupe de clients.
- Méthodes:
  - Les méthodes get et set pour les attributs.

CLASSE CONTRAT

- Attributs:
  - type: Le type de contrat de l'employé.
  - debut: La date de début du contrat.
  - fin: La date de fin du contrat.
- Méthodes:
  - estValide(Date): Vérifie si le contrat est valide à une date donnée.
  - Les méthodes get et set pour les attributs.
  - prolongerContrat(Date): Permet de prolonger la date de fin du contrat.
  - terminerContrat(): Met fin au contrat à la date actuelle.
  - toString(): Renvoie une représentation sous forme de chaîne du contrat.

CLASSE SHIFT

- Attributs:
  - heureDebut: L'heure de début du shift.
  - heureFin: L'heure de fin du shift.
  - employe: L'employé associé au shift.
- Méthodes:
  - Les méthodes get et set pour les attributs.

CLASSE HORAIRES

- Attributs:
  - horaireEmployes: Une carte associant chaque employé à une liste de ses shifts.
- Méthodes:
  - planifierEmployes(Employe, Shift): Planifie un employé pour un shift spécifique.
  - verifierDisponibilite(Employe, Date): Vérifie la disponibilité d'un employé pour un shift donné.
  - Les méthodes get et set pour l'attribut horaireEmployes.- **ajouterShift(Scanner, Restaurant), voirShifts(Scanner, Restaurant)**: Gestion des shifts des employés.

CLASSE TABLE

- Attributs:
  - numero: Le numéro de la table.
  - capacite: La capacité de la table.
  - serveurAssigne: Le serveur assigné à la table.
  - estOccupee: Un booléen indiquant si la table est occupée.
  - commande: La commande associée à la table.
- Méthodes:
  - Les méthodes get et set pour les attributs.
  - trouverTableParNumero(int, Restaurant): Trouve une table par son numéro dans un restaurant.

CLASSE TRANSACTION

- Attributs:
  - table: La table associée à la transaction.
  - serveur: Le serveur gérant la transaction.
  - commande: La commande associée à la transaction.
  - facture: La facture générée pour la transaction.
- Méthodes:
  - initierTransaction(Client): Initie une transaction pour un client.
  - effectuerPaiement(double): Effectue un paiement pour la commande.
  - terminerTransaction(): Termine la transaction une fois le paiement complet.
  - calculerTotal(): Calcule le total à payer pour la commande.
  - enregistrerFacture(): Enregistre la facture de la transaction.
  - Les méthodes get et set pour les attributs.

CLASSE COMMANDE

- Attributs:
  - plats: Liste des plats commandés.
  - boissons: Liste des boissons commandées.
  - total: Le total de la commande.
  - estPrete: Indique si la commande est prête.
- - estPreteBoissons: Indique si les boissons de la commandes sont prêtes.

- - estPretePlats: Indique si les plats de la commandes sont prêtes
  - enPreparation: Indique si la commande est en préparation.
  - montantPaye: Le montant déjà payé pour la commande.
- Méthodes:
  - effectuerPaiement(double): Permet au client de payer une partie ou la totalité de la commande.
  - estEntierementPayee(): Vérifie si la commande a été entièrement payée.
  - getMontantPaye(): Retourne le montant déjà payé.
  - marquerCommePrete(): Marque la commande comme étant prête.
- - getServeur(Restaurant restaurant) : serveur assigné à la commande
  - estPretee(): Vérifie si la commande est prête.
- -estPreteBoissons() : vérifie si les boissons sont prêtes pour cette commande
  - resumerCommande(): Résume la commande pour l'affichage.
  - calculerTotal(): Calcule le total de la commande.
  - ajouterPlat(Plat), ajouterBoisson(Boisson): Ajoute un plat ou une boisson à la commande.
  - Les méthodes get et set pour les attributs.

CLASSE FACTURE

- Attributs:
- serveurNom : nom du serveur
- tableNumero: numéro de table
- total : prix total de la transaction
- plats : liste des plats
- boissons : liste des boissons
- montantPaye : montant payé par le client
- dateFacture: date de la facture
- Méthodes:
  - générerFacture(): Génère une représentation textuelle de la facture.
  - Les méthodes get et set pour les attributs.

DIAGRAMME DE SÉQUENCE

Le diagramme de séquence décrit l'interaction entre les objets en fonction du temps dans le système de gestion de restaurant. Voici les étapes clés :

1. **Accueil des clients et assignation à une table :** Interaction du client avec l'interface utilisateur, suivi de l'accueil par le serveur et l'assignation d'une table.
2. **Prise de commande :** Le serveur prend la commande des clients et transmet les détails à la cuisine et au bar pour préparation.
3. **Préparation de la commande :** Les cuisiniers et le barman préparent les plats et les boissons.
4. **Service et paiement :** Le serveur sert la commande et génère la facture. Les clients effectuent le paiement.
5. **Gestion des stocks et des employés :** Le manager gère les stocks et planifie les horaires des employés.
6. **Clôture de la journée :** Le manager demande un rapport de performance et imprime une liste de courses pour le réapprovisionnement des stocks.

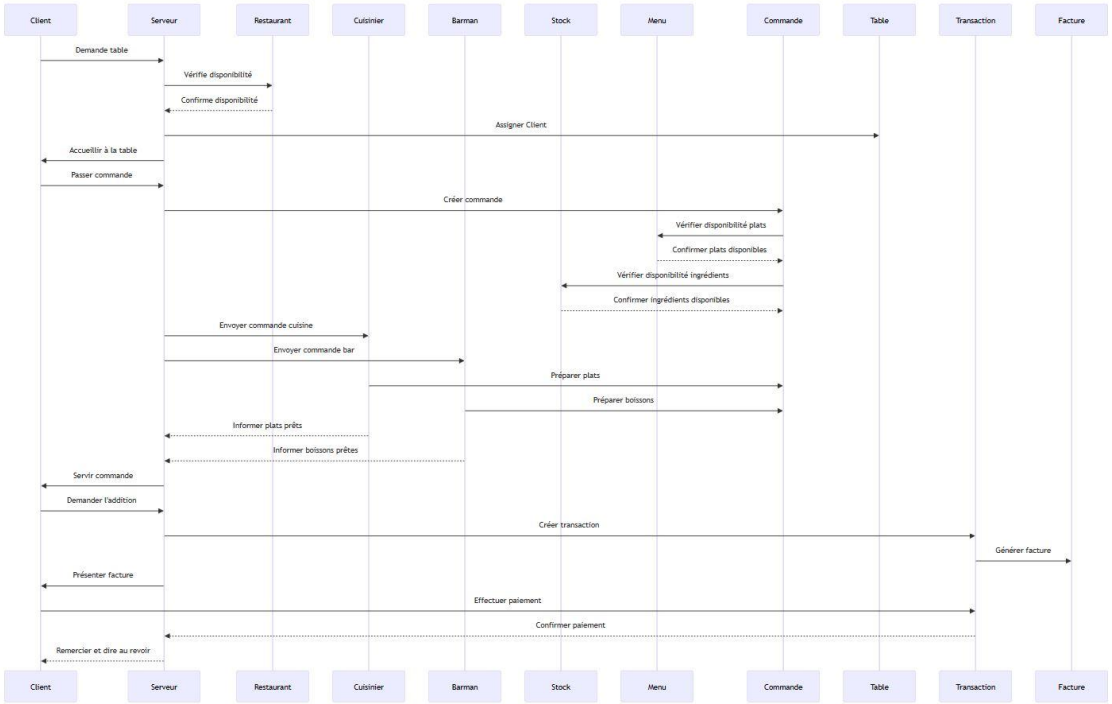


Diagramme de séquence : “Diagramme de séquence.Jpg”

Lien du code : [Edit Diagram (redstarplugin.com)]

Lien du diagramme de séquence:Online FlowChart & Diagrams Editor - Mermaid Live Editor