1

Le modèle MVC (Modèle-Vue-Contrôleur) est une architecture logicielle qui sépare une application en trois composantes principales : le Modèle, la Vue et le Contrôleur. Dans le contexte de votre simulation de stationnement, voici comment ces composantes s'appliquent spécifiquement à un véhicule.

**1. Modèle (Model)**

Le **Modèle** représente la logique métier et les données de l'application. Dans le cas d'un véhicule, cela inclut :

* **Attributs du véhicule** :
  + **vehicleLabel** : Un objet JLabel qui représente visuellement le véhicule dans l'interface utilisateur.
  + **isParked** : Un booléen qui indique si le véhicule est actuellement garé ou non.
  + **positionX et positionY** : Les coordonnées actuelles du véhicule sur l'interface graphique, permettant de gérer sa position lors du déplacement.
* **Méthodes** :
  + **parkIn(ParkingSpot spot)** : Permet au véhicule de se garer dans un emplacement spécifique. Cette méthode modifie l'état du véhicule et de l'emplacement de stationnement.
  + **move(int x, int y)** : Gère le déplacement du véhicule en mettant à jour ses coordonnées.
  + **getPosition()** : Retourne la position actuelle du véhicule.
  + **setPosition(int x, int y)** : Met à jour la position du véhicule selon les mouvements effectués.

**2. Vue (View)**

La **Vue** est responsable de l'affichage des données au utilisateur et de la présentation de l'interface. Pour un véhicule, cela inclut :

* **Représentation graphique** :
  + Le JLabel associé au véhicule qui affiche une image ou une icône représentant le véhicule. La vue ne contient pas de logique métier, mais elle affiche l'état actuel du modèle (par exemple, si le véhicule est garé ou en mouvement).
* **Mise à jour de l'affichage** :
  + La vue doit être informée des changements dans le modèle (comme le fait qu'un véhicule est garé ou déplacé) afin de mettre à jour l'affichage. Cela pourrait être réalisé par des méthodes dans la classe ParkingSimulation qui invoquent la mise à jour des propriétés de la vue.

**3. Contrôleur (Controller)**

Le **Contrôleur** gère les interactions de l'utilisateur et fait le lien entre le modèle et la vue. Dans le cas d’un véhicule, cela inclut :

* **Gestion des événements** :
  + Le CarKeyListener est un exemple de contrôleur qui écoute les événements de clavier. Lorsqu'une touche fléchée est pressée, le contrôleur va appeler les méthodes appropriées sur le modèle Vehicle pour déplacer le véhicule.
* **Logique de sélection et de parking** :
  + Lorsqu'un utilisateur sélectionne un véhicule pour le garer, le contrôleur doit interagir avec le modèle pour déterminer si le parking est possible (par exemple, vérifier si l'emplacement est déjà occupé) et mettre à jour l'état du véhicule et de l'emplacement.

**Conclusion**

En résumé, le modèle MVC pour un véhicule dans la simulation de stationnement se compose de :

* **Modèle** : Gère les données et la logique de stationnement du véhicule.
* **Vue** : Représente visuellement le véhicule sur l'interface, affichant son état et sa position.
* **Contrôleur** : Interagit avec l'utilisateur, gérant les événements de sélection et de mouvement du véhicule, tout en mettant à jour le modèle et la vue en conséquence.

2

Voici une liste des objets graphiques utilisés dans votre simulation de stationnement ainsi que les événements associés qui assurent l'interaction de l'utilisateur avec l'application.

**Objets Graphiques Utilisés**

1. **JFrame**
   * **Description** : La fenêtre principale de l'application.
   * **Événements Associés** : Aucun événement direct, mais sert de conteneur pour tous les autres composants.
2. **JPanel**
   * **Description** : Un conteneur utilisé pour organiser les boutons de stationnement dans une grille.
   * **Événements Associés** : Aucun événement direct, mais il est utilisé pour structurer l'interface utilisateur.
3. **JButton**
   * **Description** : Boutons représentant chaque emplacement de stationnement (e.g., "Spot 1", "Spot 2", etc.).
   * **Événements Associés** :
     + **ActionListener** : Lorsqu'un utilisateur clique sur un bouton, cela déclenche l'action de garer un véhicule dans cet emplacement via la méthode parkVehicle(spotIndex).
4. **JLabel**
   * **Description** : Étiquettes pour afficher le nombre de véhicules, le temps écoulé, et les messages de statut.
   * **Événements Associés** : Aucun événement direct, mais elles sont mises à jour en réponse à des actions (e.g., changement de statut, mise à jour du timer).
5. **JTextField**
   * **Description** : Champ de texte pour entrer le nombre de véhicules que l'utilisateur souhaite simuler.
   * **Événements Associés** :
     + **ActionListener** : Lorsqu'un utilisateur appuie sur "Entrer" ou après avoir saisi un nombre, il peut déclencher la simulation (par exemple, en cliquant sur le bouton "Start").
6. **JButton (Start and Reset)**
   * **Description** : Boutons pour démarrer la simulation et réinitialiser l'état.
   * **Événements Associés** :
     + **ActionListener** : Le bouton "Start" initialise la simulation avec le nombre de véhicules spécifié, et le bouton "Reset" remet à zéro la simulation.
7. **ImageIcon** (pour les véhicules)
   * **Description** : Utilisé pour représenter visuellement les véhicules dans l'interface.
   * **Événements Associés** : Aucun événement direct, mais l'image est affichée sur les JLabel représentant les véhicules.
8. **CarKeyListener**
   * **Description** : Un écouteur de clavier qui permet de déplacer le véhicule sélectionné avec les touches fléchées.
   * **Événements Associés** :
     + **KeyListener** : Lorsqu'une touche fléchée est pressée (haut, bas, gauche, droite), cela déclenche des mouvements du véhicule sélectionné à l'écran.

**Résumé des Interactions**

* **Sélection de Véhicule** : L'utilisateur clique sur un JLabel représentant un véhicule, ce qui met à jour la variable selectedVehicle.
* **Garer un Véhicule** : L'utilisateur clique sur un JButton associé à un emplacement de stationnement, ce qui appelle la méthode parkVehicle(spotIndex).
* **Démarrer la Simulation** : L'utilisateur entre un nombre dans le JTextField et clique sur le bouton "Start", initiant la simulation.
* **Réinitialiser la Simulation** : Cliquer sur le bouton "Reset" remet à zéro l'état de la simulation.
* **Déplacer le Véhicule** : Utiliser les touches fléchées pour déplacer le véhicule sélectionné à l'écran.

Ces objets graphiques et événements associés permettent une interaction fluide et intuitive avec l'application, facilitant la simulation de stationnement pour l'utilisateur.

3

Voici une justification des choix de styles pour l'interface utilisateur de la simulation de stationnement, ainsi qu'un rappel des recommandations ergonomiques suivies.

**Justification des Choix de Styles**

1. **Couleurs**:
   * **Vert pour les emplacements libres** : Utilisé pour indiquer les places de stationnement disponibles. Le vert est associé à la sécurité et à l'accessibilité, ce qui aide les utilisateurs à identifier facilement où ils peuvent garer leurs véhicules.
   * **Rouge pour les emplacements occupés** : Ce choix de couleur signale visuellement que ces places ne sont plus disponibles. Le rouge attire l'attention et est souvent associé à l'interdiction, renforçant la compréhension immédiate de l'état des emplacements.
   * **Bleu pour les véhicules** : Cette couleur est apaisante et distincte, permettant aux utilisateurs de reconnaître facilement les véhicules sur l'interface.
2. **Typographie**:
   * **Police Arial, taille 16-28** : Arial est une police sans-serif facile à lire et largement utilisée dans les interfaces utilisateur. La taille des polices est choisie pour assurer une bonne lisibilité, même à distance, ce qui est essentiel pour une expérience utilisateur positive.
3. **Disposition (Layout)**:
   * **GridLayout pour les emplacements de stationnement** : Utiliser une grille pour organiser les boutons de stationnement reflète la manière dont les voitures sont généralement garées. Cela crée une analogie intuitive et facilite la navigation.
   * **Positionnement des composants** : Les éléments essentiels, comme le champ de saisie pour le nombre de véhicules et les boutons "Start" et "Reset", sont placés de manière à ce qu'ils soient facilement accessibles et visibles, réduisant ainsi le temps de recherche de l'utilisateur.
4. **Feedback Visuel**:
   * **Changements de couleur et de texte** : Les boutons changent de couleur et de texte lorsqu'ils sont occupés, fournissant un retour immédiat à l'utilisateur sur l'état de l'emplacement. Cela renforce la compréhension et l'interaction.

**Recommandations Ergonomiques Suivies**

1. **Simplicité**:
   * L'interface doit rester simple et claire, sans éléments superflus. Cela permet aux utilisateurs de se concentrer sur l'essentiel : garer et déplacer les véhicules.
2. **Clarté**:
   * Les labels et les boutons sont étiquetés de manière explicite (par exemple, "Spot 1", "Start", "Reset"), ce qui aide les utilisateurs à comprendre rapidement leurs fonctions.
3. **Consistance**:
   * Les styles de boutons, polices et couleurs sont utilisés de manière cohérente à travers l'application. Cela crée une expérience utilisateur fluide et prévisible, facilitant l'apprentissage et l'utilisation de l'application.
4. **Accessibilité**:
   * Les tailles de texte et les contrastes de couleurs sont choisis pour garantir la lisibilité, même pour les utilisateurs ayant des problèmes de vision. Les boutons sont suffisamment grands pour être cliqués facilement, même sur des écrans tactiles.
5. **Feedback**:
   * L'interface offre un retour immédiat sur les actions des utilisateurs (par exemple, lorsque les véhicules sont garés ou lorsque les spots sont occupés), ce qui améliore l'interaction et réduit les frustrations.
6. **Aide à la Navigation**:
   * Les utilisateurs peuvent facilement naviguer dans l'application grâce à la disposition intuitive et aux éléments interactifs bien définis. Les événements associés (comme les clics et les pressions de touches) sont clairement liés à des actions visibles.

**Conclusion**

Les choix de styles et les recommandations ergonomiques appliqués à cette simulation de stationnement visent à créer une interface conviviale, accessible et efficace. Ces éléments contribuent à une meilleure expérience utilisateur, facilitant la compréhension et l'interaction avec l'application.

Expliquation de code :

Voici une explication détaillée du code de la classe ParkingSimulation, qui est une application graphique Java utilisant Swing pour simuler un système de stationnement interactif.

**Structure et Fonctionnalités de la Classe**

1. **Déclaration des Attributs** :
   * vehicleCount: Compte le nombre de véhicules à garer.
   * parkingSpots: Un tableau de 10 éléments qui indique l'état (occupé ou libre) de chaque emplacement de stationnement.
   * spotButtons: Tableau de boutons représentant chaque emplacement de stationnement.
   * vehicles: Un tableau contenant des objets JLabel représentant les véhicules.
   * selectedVehicle: Indique l'index du véhicule sélectionné.
   * timer: Un objet Timer pour suivre le temps écoulé.
   * timerLabel et statusLabel: Labels pour afficher le temps et les messages d'état.
   * startTime: Timestamp pour mesurer le temps écoulé.
   * timerRunning: Booléen pour indiquer si le timer est en cours d'exécution.
   * parkedVehicles: Liste qui indique si chaque véhicule est garé.
   * carImage: Image utilisée pour représenter les véhicules.
2. **Constructeur de la Classe** :
   * **Initialisation de la Fenêtre** : Définit le titre, l'opération de fermeture, la taille, la disposition, et la couleur de fond de la fenêtre principale.
   * **Chargement de l'Image** : Charge une image de voiture à partir d'un chemin spécifié.
   * **Appel à initUIComponents()** : Méthode pour initialiser les composants de l'interface utilisateur.
   * **Ajout de l'Écouteur de Clavier** : Permet de capturer les événements de clavier pour déplacer les véhicules.
3. **Méthode initUIComponents()** :
   * **Création de l'En-tête** : Un JLabel centré pour afficher le titre de l'application.
   * **Création de la Grille de Stationnement** : Un JPanel en grille contenant des boutons pour chaque emplacement de stationnement. Chaque bouton est configuré pour appeler parkVehicle() lorsqu'il est cliqué.
   * **Labels pour le Temps et le Statut** : Labels pour afficher le temps écoulé et le message d'état.
   * **Contrôles Inférieurs** : Comprend un champ de texte pour entrer le nombre de véhicules, et des boutons "Start" et "Reset". Le bouton "Start" commence la simulation après validation de l'entrée, et le bouton "Reset" remet à zéro la simulation.
4. **Méthode startSimulation(int count)** :
   * Initialise le nombre de véhicules et crée des JLabel pour chaque véhicule avec l'image de la voiture.
   * Ajoute des écouteurs de souris pour sélectionner un véhicule en cliquant dessus.
   * Réinitialise le timer lorsque la simulation commence.
5. **Méthodes de Timer** :
   * **resetTimer()** : Réinitialise le timer et le remet à zéro.
   * **startTimer()** : Démarre le timer si ce n'est pas déjà fait.
   * **updateTimer()** : Met à jour le label du timer avec le temps écoulé depuis le début de la simulation.
6. **Méthode resetSimulation()** :
   * Réinitialise tous les composants de l'application à leur état par défaut, y compris les emplacements de stationnement et les véhicules.
7. **Méthode parkVehicle(int spotIndex)** :
   * Gère le processus de stationnement d'un véhicule dans l'emplacement sélectionné.
   * Vérifie si l'emplacement est déjà occupé ou si le véhicule sélectionné est déjà garé.
   * Met à jour l'état du véhicule et de l'emplacement dans l'interface en modifiant les couleurs et les textes des boutons.
8. **Méthode checkAllParked()** :
   * Vérifie si tous les véhicules sont garés. Si oui, elle arrête le timer et affiche un message de succès.
9. **Classe Interne CarKeyListener** :
   * Écoute les événements de clavier pour déplacer le véhicule sélectionné. Les déplacements sont limités par les bords de la fenêtre pour éviter que le véhicule ne sorte de l'écran.
10. **Méthode main** :
    * Point d'entrée de l'application. Utilise SwingUtilities.invokeLater pour garantir que l'interface utilisateur est créée sur le thread de l'interface graphique.

**Conclusion**

La classe ParkingSimulation est une application interactive qui permet aux utilisateurs de simuler le stationnement de véhicules. Elle utilise des composants graphiques Swing pour créer une interface conviviale, gère les interactions avec les utilisateurs à travers des événements et offre une expérience de simulation dynamique grâce à la gestion du temps et de l'état des véhicules.