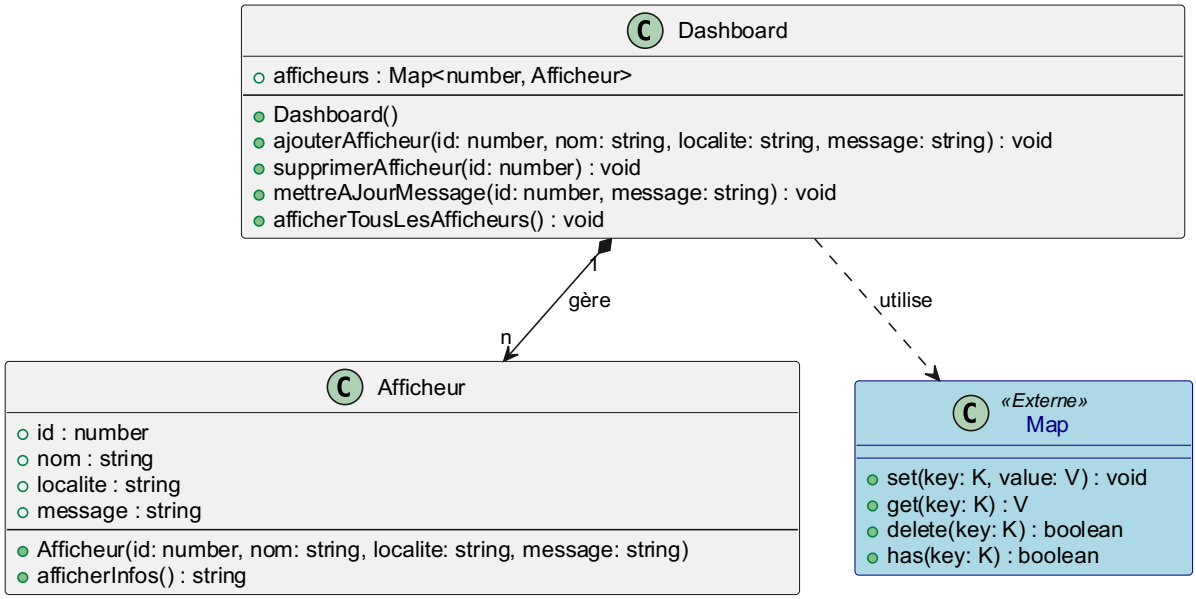
DASHBOARD GESTION CENTRALISÉE AFFICHEURS MÉTRO

Diagramme de classe :



Aperçu du dashboard :

QUESTIONS :

1. Indiquez la visibilité des champs et des méthodes.
2. Indiquez la nature de la liaison entre les deux classes.

Classe Afficheur :

1. Créez une classe Afficheur avec les propriétés suivantes :
   - id (identifiant unique)
   - nom (nom de l'afficheur)
   - localite (localité de l'afficheur)
   - message (message affiché).
2. Ajoutez un constructeur dans la classe pour initialiser ces propriétés.
3. Ajoutez une méthode afficherInfos() dans la classe qui retourne une chaîne contenant les informations de l'afficheur.

4. Tester le classe avec le code suivant :

```
const monAfficheur = new Afficheur(1, "Afficheur Test", "Station A", "Bienvenue !");
console.log(monAfficheur.afficherInfos());

// Résultat attendu :
// Afficheur Afficheur Test (1) – Localité: Station A, Message: "Bienvenue !"
```

Classe Dashboard :

1. Créez une classe Dashboard avec une propriété afficheurs qui stocke une liste (ou un Map) d'afficheurs.
2. Ajoutez une méthode ajouterAfficheur(id, nom, localite, message) pour ajouter un afficheur.
   - Vérifiez qu'un afficheur avec le même ID n'existe pas déjà.
3. Ajoutez une méthode supprimerAfficheur(id) pour supprimer un afficheur par son ID.
4. Ajoutez une méthode afficherTousLesAfficheurs() qui affiche toutes les informations des afficheurs dans la console.
   - Utiliser une boucle forEach().

5. Tester la classe Dashboard :

```
const dashboard = new Dashboard();
dashboard.ajouterAfficheur(1, "Afficheur 1", "Station A", "Message 1");
dashboard.ajouterAfficheur(2, "Afficheur 2", "Station B", "Message 2");
dashboard.afficherTousLesAfficheurs();

// Résultat attendu :
// Afficheur Afficheur 1 (1) – Localité: Station A, Message: "Message 1"
// Afficheur Afficheur 2 (2) – Localité: Station B, Message: "Message 2"
```

Intégration du HTML

1. Écrivez un fichier HTML contenant :

- Un formulaire avec des champs pour id, nom, localite et message.
- Un bouton pour soumettre le formulaire.
- Une section où les afficheurs seront listés dynamiquement.

2. Ajoutez un script JavaScript pour capturer les valeurs du formulaire (via addEventListener) et afficher les valeurs dans la console.

3. Modifiez la méthode afficherTousLesAfficheurs() pour qu'elle affiche les informations des afficheurs dans la section dédiée du HTML.

4. Ajoutez un bouton "Supprimer" pour chaque afficheur.

5. Testez le code pour ajouter et supprimer des afficheurs dynamiquement.

6. Ajoutez une méthode dans la classe Dashboard pour mettre à jour le message d'un afficheur.

7. Modifiez l'affichage HTML pour inclure un champ de texte et un bouton "Mettre à jour" pour chaque afficheur.

8. Test l'application en vérifiant que l'application permet d'ajouter, de supprimer et de modifier les afficheurs.

Bonus : Questions pour aller plus loin

1. Comment pourriez-vous enregistrer les afficheurs pour qu'ils soient disponibles après avoir fermé la page ? (Introduction au stockage local avec localStorage).
2. Comment pourriez-vous envoyer les données des afficheurs vers un serveur ? (Requêtes HTTP avec fetch).
3. Comment pourriez-vous faire une mise à jour en temps réel sans recharger la page ? (Introduction aux WebSockets ou à l'actualisation dynamique).

ANNEXE : LA CLASSE MAP

La méthode new Map() en JavaScript crée une nouvelle instance de la classe Map, qui est une structure de données permettant de stocker des paires clé-valeur. Les objets Map offrent des fonctionnalités similaires aux objets ordinaires, mais avec des avantages supplémentaires pour gérer des collections de données.

Caractéristiques de Map

1. **Clés uniques** :
   - Chaque clé dans une instance de Map est unique.
   - Les clés peuvent être **de n'importe quel type**, y compris des objets, des fonctions, ou des primitives comme les nombres ou les chaînes.
2. **Ordre d'insertion** :
   - Les éléments d'une Map sont itérés dans l'ordre où ils ont été insérés.
3. **Taille** :
   - La taille d'une Map est accessible via la propriété size.

```
const map = new Map();
```

Principales Méthodes

Méthode	Description
set(key, value)	Ajoute ou met à jour un élément avec la clé key et la valeur value.
get(key)	Récupère la valeur associée à la clé key.
delete(key)	Supprime l'élément associé à la clé key.
has(key)	Vérifie si une clé existe dans la Map. Renvoie true ou false.
clear()	Supprime tous les éléments de la Map.
keys()	Renvoie un itérateur pour les clés.
values()	Renvoie un itérateur pour les valeurs.
entries()	Renvoie un itérateur pour les paires [clé, valeur].

Exemple d'Utilisation

```
// Créer une Map vide
const map = new Map();

// Ajouter des paires clé-valeur
map.set("nom", "Alice");
map.set("age", 25);
map.set("ville", "Paris");

// Obtenir une valeur
console.log(map.get("nom")); // Alice

// Vérifier si une clé existe
console.log(map.has("age")); // true
console.log(map.has("pays")); // false

// Supprimer un élément
map.delete("ville");
console.log(map.has("ville")); // false

// Parcourir les éléments
for (const [key, value] of map.entries()) {
    console.log(`${key}: ${value}`);
}
```

```
// Résultat:
// nom: Alice
// age: 25

// Obtenir la taille
console.log(map.size); // 2

// Vider la Map
map.clear();
console.log(map.size); // 0
```