

# Programmation Orientée Objet Java

## Cas Navire

### Objectifs du TP :

- ✓ **Ajouter des nouvelles fonctionnalités au TP précédent en introduisant de nouveaux concepts**
- ✓ **Introduire l'héritage**
- ✓ S'initier à la programmation objet en Java
- ✓ Utiliser les bonnes pratiques pour le débogage
- ✓ Lire une documentation et savoir l'appliquer
- ✓ Réviser les concepts algorithmiques
- ✓ Se forcer à se poser des questions et en obtenir des réponses
- ✓ Sensibiliser au développement par la méthode des tests

Ce TP n'a pas vocation à être un TP professionnel. Il est seulement péda

### Prérequis :

- ✓ **Avoir fini le TP précédent**
- ✓ Bases algorithmiques
- ✓ Bases du langage Java
- ✓ Bases théoriques de la POO dont l'héritage et les interfaces
- ✓ Principales fonctionnalités de l'IDE Eclipse

### Dans ce TP, vous serez tour à tour :

- ✓ Concepteur de classe :



- ✓ Utilisateur de classe :



## Partie 1 : MISE EN PLACE DU PROJET

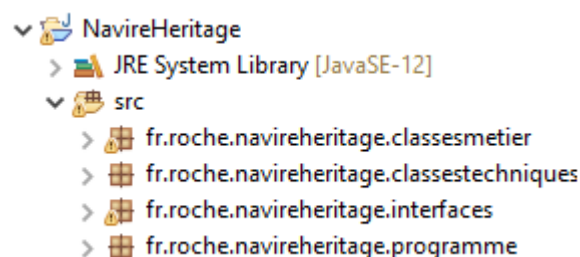


Il vous est demandé de créer un nouveau projet java pour éviter les effets de bord dans la mise au point.

Ce projet s'appellera NavireHeritage

Chaque classe sera implémentée dans un package.

Je vous propose cette structure :



La racine du package sera :

**fr.votrenom.navireheritage**

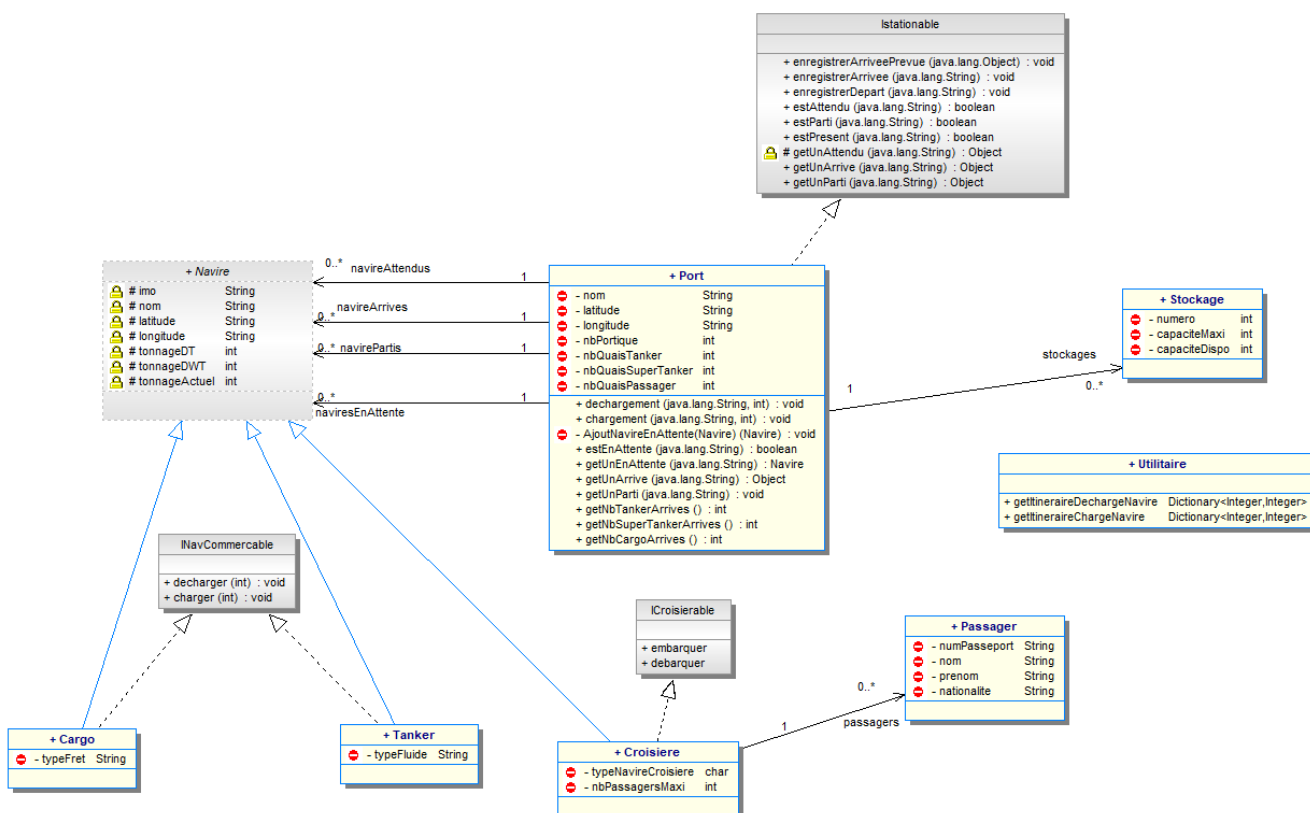
puis :

- ✓ **fr.votrenom. navireheritage.classesmetier** : pour stocker les classes métier
- ✓ **fr.votrenom. navireheritage.programme** : pour stocker la classe Programme
- ✓ **fr.votrenom. navireheritage.interfaces** : pour stocker les interfaces nécessaires à certaines classes métier.
- ✓ **fr.votrenom. navireheritage.classestechnique** : pour stocker les classes techniques nécessaires à l'application.

## Partie 2 : DIAGRAMME DE CLASSES

Par rapport au TD précédent, vous allez gérer un port pouvant accueillir 3 types différents de navires :

- ✓ Les navires cargo : Le fret est toujours constitué de containers. il faudra être capable de charger et décharger des containers dans des zones de stockages répertoriées.
- ✓ Les navires tanker : ils transportent des fluides (huile, pétrole, gaz liquide,...) . Il faudra être capable de charger et décharger des volumes de fluide.
- ✓ Les navires passagers : ils transportent donc des croisiéristes, il faudra gérer les embarquements et débarquements de personnes.



Pour assurer la gestion du port, le diagramme de classes suivant a été conçu :

La description des classes et des interfaces vous est fournie dans l'annexe 1.



---

### Travail à faire

Justifiez pourquoi la classe Navire est une classe abstraite.  
Portez une réflexion à ce sujet

Réponse collégiale

---



Une méthode formatée d'une classe métier ne doit pas excéder 20 lignes hors commentaires.

Vous serez donc amenés à ajouter, dans les classes, des méthodes privées pour réaliser certains traitements et rendre moins complexe la lecture d'autres méthodes.



**Tous les contrôles mis en place dans les précédents TD doivent être reconduits, sauf avis contraire**

**Vous recréerez également la classe GestionPortException dans le package fr.roche.navire.exceptions. Cette classe héritera de la classe Exception**

Les messages d'erreur s'afficheront sur 2 lignes :

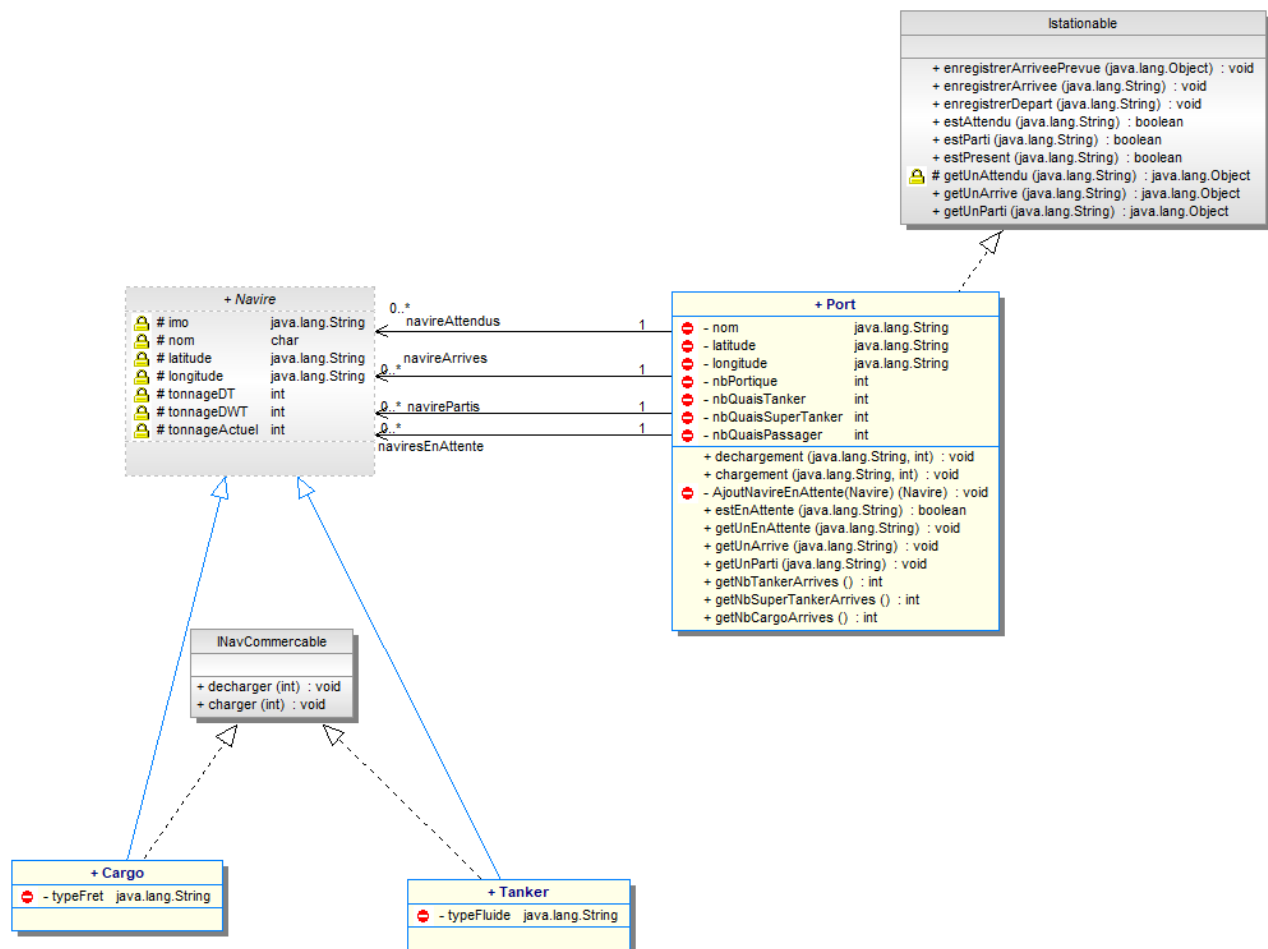
```
Erreur le 07 févr. 2020 à 15:05:16  
message erreur à personnaliser
```

### Partie 3 : CLASSES PORT ET NAVIRE

On vous demande dans cette partie d'implémenter les classes

- ✓ Port
- ✓ Navire
- ✓ Cargo
- ✓ Tanker

Et les interfaces nécessaires. Diagramme UML correspondant :



## 1. Initialisation du port

### Travail à faire



Créer les classes et les interfaces

Pour chaque classe, créez une méthode toString qui renvoie un résultat similaire aux exemples fournis dans le sujet.

Testez votre travail avec les exemples fournis par le professeur  
Le code suivant dans la méthode Main

```
Port port = new Port("Marseille", "43.2976N", "5.3471E", 4, 3, 2,4);
System.out.println(port);
```

```
public class Programme {

    public static void main(String[] args) {
        try {
            Port port = new Port("Marseille", "43.2976N", "5.3471E", 4, 3, 2, 4);
            System.out.println(port);
        }
    }
}
```

Doit fournir le résultat suivant :

```
-----
Port de Marseille
Coordonnées GPS : 43.2976N / 5.3471E
Nb portiques : 4
Nb quais croisière : 4
Nb quais tankers : 3
Nb quais super tankers : 2
Nb Navires à quai : 0
Nb Navires attendus : 0
Nb Navires à partis : 0
Nb Navires en attente : 0

Nombre de cargos dans le port : 0
Nombre de tankers dans le port : 0
Nmbre de super tankers dans le port : 0
-----
```



### Travail à faire

Vous allez créer une classe abstraite Test qui va vous permettre d'implémenter les méthodes permettant de tester tous les cas de figure de l'application.

Vous commencerez par créer la méthode statique chargementInitial() qui permettra de tester la méthode EnregistrerArriverPrevue de la classe Port. Vous chargerez quelques navires à partir du fichier *chargementInitial.txt* mis à disposition sur moodle

Code de la méthode main :

```
public static void main(String[] args) {
    try {
        Port port = new Port("Marseille", "43.2976N", "5.3471E", 4, 3, 2,4);
        Test.chargementInitial(port);
        System.out.println(port);
        Test.afficheAttendus(port);
    }
}
```

Extrait de la méthode chargementInitial de la classe Port :

Vous devriez obtenir ceci :

```
-----
public abstract class Test {
    public static void chargementInitial(Port port) {
        try {
            // cargos
            port.enregistrerArriveePrevue(new Cargo("IMO9780859", "CMA CGM A. LINCOLN", "43.43279 N", "134.76258 W",
                140872, 148992, 123000, "marchandises diverses"));
                Nb Navires à quai : 0
                Nb Navires attendus : 11
                Nb Navires à partis : 0
                Nb Navires en attente : 0

            Nombre de cargos dans le port : 0
            Nombre de tankers dans le port : 0
            Nmbre de super tankers dans le port : 0
            -----
            Liste des bateaux en attente de leur arrivée :
            IMO9220952    HARAD : Tanker
            IMO9780859    CMA CGM A. LINCOLN : Cargo
            IMO9241061    RMS QUEEN MARY 2 : Croisiere
            IMO9197832    KALAMOS : Tanker
            IMO9803613    MSC GRANDIOSA : Croisiere
            IMO9250098    CONTAINERSHIPS VII : Cargo
            IMO9380374    CITRUS : Tanker
            IMO9379715    NEW DRAGON : Tanker
            IMO9502910    MAERSK EMERALD : Cargo
            IMO9351476    CRUISE ROMA : Croisiere
            IMO9334076    EJNAN : Tanker
        }
    }
}
```



### Travail à faire

Créez dans la classe Test la méthode statique testEnregistrerArriveePrevue (Port port, Navire navire). Cette méthode accepte en paramètre le navire .

```
public static void testEnregistrerArriveePrevue(Port port, Navire navire) {
    try {
        port.enregistrerArriveePrevue(navire);
    } catch (Exception ex) {
        System.out.println(ex.getMessage());
    }
}
```

A l'exécution :

```
Test.testEnregistrerArriveePrevue(port, new Cargo("IM09780859", "CMA CGM A. LINCOLN", "43.43279 N", "134.76258 W",
140872,148992,123000, "marchandises diverses"));
```

Le navire IM09780859 est déjà enregistré dans les navires attendus

Toutes vos méthodes de test seront construites sur le modèle de la méthode testEnregistrerArriveePrevue (...)

Elles ne serviront qu'à tester UNE fonctionnalité.

## 2. Gestion de l'arrivée d'un navire attendu

### Processus d'arrivée dans un port d'un navire dans le port :

Le port reçoit en temps réel les informations concernant un navire. Ainsi, les étapes de la vie d'un Navire dans un port sont :

Navire Attendu => Navire Arrivé => Navire parti

Ou

Navire Attendu => Navire En Attente => Navire Arrivé => Navire parti

Ce dernier cas ne concerne que les navires de commerce (Cargo et tanker), et ils sont mis en attente lors de leur arrivée si il n'y a pas de quai adéquate pour les accueillir.

Dans tous les cas, on ne peut enregistrer l'arrivée d'un navire non attendu. Si on essaie d'enregistrer l'arrivée d'un navire non attendu, il y a déclenchement d'une Exception.

Navire attendu croisière	Il y a toujours de la place.	à l'aide de son Id (imo), on l'enlève du dictionnaire des navires attendus et on l'enregistre dans la collection des navires arrivés
Navire attendu (cargo ou tanker)	Il y a de la place pour son type	à l'aide de son Id (imo), on l'enlève du dictionnaire des navires attendus et on l'enregistre dans la collection des navires arrivés
	Il n'y a pas de la place pour son type	à l'aide de son Id (imo), on l'enlève du dictionnaire des navires attendus et on

		l'enregistre dans la collection des navires en attente. Ils seront toujours prioritaires pour l'enregistrement de l'arrivée dans le cas d'un départ. Si c'est un tanker, on incrémente la
Navire non attendu (cargo ou tanker)	Il y a de la place pour son type	On l'enregistre directement dans le dictionnaire des navires arrivés après avoir créé un objet Navire.
	Il n'y a pas de la place pour son type	On l'enregistre directement dans le dictionnaire des navires en attente.



Attention, il y a deux types de quais pour les tankers. Il faut placer le bon navire tanker sur le bon quai.

Un navire commercial réserve toujours longtemps à l'avance un quai dans le port. Il ne peut donc être mis en attente.

Test de l'arrivée d'un Navire de type Croisière :



Je vous conseille de créer des méthodes privées pour alléger l'écriture de la méthode enregistrerArrivee de la classe port

Dans la classe Test :

```
public static void testEnregistrerArrivee(Port port, String imo) {
    try {
        port.enregistrerArrivee(imo);
        System.out.println("navire " + imo + " arrivé");
    } catch (Exception ex) {
        System.out.println(ex.getMessage());
    }
}
```

Dans le Main :

```
/**
 * enregistrement de l'arrivée d'un Navire de la classe Croisiere
 * Il y a toujours de la place.
 */
Test.testEnregistrerArrivee(port, "IM09241061");
```

Résultat :

navire IM09241061 arrivé

Dans le code suivant, on va essayer d'enregistrer les arrivées :

- ✓ D'un navire non attendu,
- ✓ D'un navire déjà arrivé :

```
Test.testEnregistrerArrivee(port, "IM00000000");
Test.testEnregistrerArrivee(port, "IM09241061");
```

Résultat d'exécution :

Le navire d'id IM00000000 n'est pas attendu ou est déjà dans le port  
Le navire d'id IM09241061 n'est pas attendu ou est déjà dans le port

Vérification à l'aide d'un point d'arrêt :

Name	Value
no method return value	
args	String[0] (id=20)
port	Port (id=22)
latitude	"43.2976N" (id=26)
longitude	"5.3471E" (id=32)
navireArrives	HashMap<K,V> (id=33)
[0]	HashMap\$Node<K,V> (id=65)
key	"IMO9241061" (id=66)
value	Croisiere (id=67)
navireAttendus	HashMap<K,V> (id=38)
[0]	HashMap\$Node<K,V> (id=51)
[1]	HashMap\$Node<K,V> (id=52)
[2]	HashMap\$Node<K,V> (id=53)
[3]	HashMap\$Node<K,V> (id=54)
[4]	HashMap\$Node<K,V> (id=55)
[5]	HashMap\$Node<K,V> (id=56)
[6]	HashMap\$Node<K,V> (id=57)
[7]	HashMap\$Node<K,V> (id=58)
[8]	HashMap\$Node<K,V> (id=59)
[9]	HashMap\$Node<K,V> (id=60)
navireEnAttente	HashMap<K,V> (id=39)
navirePartis	HashMap<K,V> (id=40)
nbPortique	4

Dans le code suivant, on va enregistrer l'arrivée de navires prévus :sans se soucier de la place disponible

- ✓ 1 tanker.
- ✓ 2 super tanker

A ce niveau, il y a suffisamment de quais pour accueillir ces 3 navires.

```
/**
 * Dans ce test, on enregistre l'arrivée d'un petit tanker attendu
 * et il y a de la place.
 */
Test.testEnregistrerArrivee(port,"IM09334076");
/**
 * On rajoute 2 super tankers qui sont attendus,
 */
Test.testEnregistrerArrivee(port,"IM09197832");
Test.testEnregistrerArrivee(port,"IM09220952");
```

Résultat :

```
navire IM09334076 arrivé
navire IM09197832 arrivé
navire IM09220952 arrivé
```

Puis on va enregistrer l'arrivée prévue d'un super tanker.

Le problème est qu'il n'y aura pas assez de quais pour l'accueillir.





Vous allez donc devoir :

- ✓ L'ajouter dans la liste des navires en attente
- ✓ Le retirer de la liste des navires attendus.

Résultat :






```
/**
 * navire IM09379715 arrivé
 * Arrivée d'un super tanker attendu , il doit être mis en attente
 */
Test.testEnregistrerArrivee(port,"IM09379715");
```

Mais, en regardant de plus près avec un point d'arrêt :

▼  navireArrives	HashMap<K,V> (id=33)	▼  navireAttendus	HashMap<K,V> (id=38)
▼  [0]	HashMap\$Node<K,V> (id=87)	▼  [0]	HashMap\$Node<K,V> (id=59)
>  key	"IMO9220952" (id=87)	>  key	"IMO9780859" (id=59)
>  value	Tanker (id=88)	>  value	Cargo (id=60)
▼  [1]	HashMap\$Node<K,V> (id=89)	▼  [1]	HashMap\$Node<K,V> (id=64)
>  key	"IMO9241061" (id=89)	>  key	"IMO9803613" (id=64)
>  value	Croisiere (id=90)	>  value	Croisiere (id=65)
▼  [2]	HashMap\$Node<K,V> (id=91)	▼  [2]	HashMap\$Node<K,V> (id=67)
>  key	"IMO9197832" (id=91)	>  key	"IMO9250098" (id=67)
>  value	Tanker (id=92)	>  value	Cargo (id=68)
▼  [3]	HashMap\$Node<K,V> (id=93)	▼  [3]	HashMap\$Node<K,V> (id=69)
>  key	"IMO9334076" (id=93)	>  key	"IMO9380374" (id=69)
>  value	Tanker (id=94)	>  value	Tanker (id=70)
		▼  [4]	HashMap\$Node<K,V> (id=72)
		>  key	"IMO9502910" (id=72)
		>  value	Cargo (id=73)
		▼  [5]	HashMap\$Node<K,V> (id=95)
		>  key	"IMO9351476" (id=95)
		>  value	Croisiere (id=96)
		▼  navireEnAttente	HashMap<K,V> (id=39)

**Il n'est pas encore dans les navires arrivés**

**Il n'est plus dans les navires attendus**

▼  navireEnAttente	HashMap<K,V> (id=3)
▼  [0]	HashMap\$Node<K,V> (id=79)
>  key	"IMO9379715" (id=79)
>  value	Tanker (id=80)
 navirePartis	HashMap<K,V> (id=4)
 nbPortique	4

**Il est dans les navires en attente**



Vous risquez de tomber sur l'erreur suivante :

class java.util.HashMap\$Values cannot be cast to class java.util.Vector

Vous pourrez avoir un élément de réponse ici :

<https://stackoverflow.com/questions/15522546/why-hashmap-values-are-not-cast-in-list>

Expliquez pourquoi vous aviez cette erreur

### 3. Gestion des départs du port

Processus de départ du port :

On retire le Navire du dictionnaire des navires arrivés grâce à son id (imo) et on l'ajoute au dictionnaire des navires partis. S'il y a des navires en attente dans la collection des navires en attente, on voit si l'un d'eux est compatible avec le quai qui s'est libéré et on enregistre son arrivée.

Pour tester les départs, vous allez créer la méthode **testEnregistrerDepart(Port port, String imo):**

```
Test.testEnregistrerDepart(port, "IM09197822"),
/**
 * On essaie de faire partir un navire qui n'est pas arrivé
 */
Test.testEnregistrerDepart(port, "IM09197822");
```

Vous allez commencer simplement. Vous ferez partir un navire qui n'est pas dans la liste des navires arrivés :

```
public static void testEnregistrerDepart(Port port, String imo) {
    try {
        port.enregistrerDepart(imo);
        System.out.println("navire " + imo + " parti");
    } catch (Exception ex) {
        System.out.println(ex.getMessage());
    }
}
```

Résultat :

```
Enregistrement départ impossible pour IM09197822 le navire n'est pas dans le port
```




















Vous allez faire partir un Navire de croisière, cela doit provoquer :

- ✓ Son ajout dans la liste des navires partis
- ✓ Sa suppression dans la liste des navires arrivés
- ✓ La recherche dans les navires en attente et éventuellement l'enregistrement de l'arrivée d'un autre super tanker

```
/**
 * On fait partir le navire de croisière,
 * il y a toujours ls super tanker en attente
 */
Test.testEnregistrerDepart(port, "IM09241061");
```

On voit

- ✓ Que le navire est dans la liste des navires partis
- ✓ qu'il n'est plus dans la liste des navires arrivés
- ✓ qu'il y a toujours le navire dans la liste des navires en attente












▼  navireArrives	HashMap<K,V> (id=33)
▼  [0]	HashMap\$Node<K,V> (id=75)
>  key	"IMO9220952" (id=78)
>  value	Tanker (id=79)
▼  [1]	HashMap\$Node<K,V> (id=76)
>  key	"IMO9197832" (id=80)
>  value	Tanker (id=81)
▼  [2]	HashMap\$Node<K,V> (id=77)
>  key	"IMO9334076" (id=82)
>  value	Tanker (id=83)
>  navireAttendus	HashMap<K,V> (id=38)
▼  navireEnAttente	HashMap<K,V> (id=39)
▼  [0]	HashMap\$Node<K,V> (id=68)
>  key	"IMO9379715" (id=69)
>  value	Tanker (id=70)
▼  navirePartis	HashMap<K,V> (id=40)
▼  [0]	HashMap\$Node<K,V> (id=51)
>  key	"IMO9241061" (id=54)
>  value	Croisiere (id=55)
▢ nbPortique	4

Vous allez maintenant faire partir le petit tanker :

- ✓ Le super tanker doit rester en attente.

```
/**
 * On fait partir le tanker,
 * le super tanker doit rester en attente
 */
Test.testEnregistrerDepart(port, "IMO9334076");
```

Constatez le résultat avec un point d'arrêt :

▼  navireEnAttente	HashMap<K,V> (id=39)
▼  [0]	HashMap\$Node<K,V> (id=65)
>  key	"IMO9379715" (id=66)
>  value	Tanker (id=67)
▼  navirePartis	HashMap<K,V> (id=40)
▼  [0]	HashMap\$Node<K,V> (id=51)
>  key	"IMO9241061" (id=60)
>  value	Croisiere (id=61)
▼  [1]	HashMap\$Node<K,V> (id=52)
>  key	"IMO9334076" (id=55)
>  value	Tanker (id=56)



























On voit :

- ✓ Que le navire est dans la liste des navires partis
- ✓ qu'il n'est plus dans la liste des navires arrivés
- ✓ qu'il y a toujours le navire dans la liste des navires en attente

Enfin, vous allez faire partir un super tanker, le navire en attente doit alors être enregistré dans la liste des navires arrivés :

```
/**
 * On fait partir le super tanker, celui en attente doit arriver
 */
Test.testEnregistrerDepart(port,"IMO9197832");
```

▼  navireArrives	HashMap<K,V> (id=33)
>  [0]	HashMap\$Node<K,V> (id=60)
▼  [1]	HashMap\$Node<K,V> (id=61)
>  key	"IMO9379715" (id=67)
>  value	Tanker (id=68)
>  navireAttendus	HashMap<K,V> (id=38)
 navireEnAttente	HashMap<K,V> (id=39)
▼  navirePartis	HashMap<K,V> (id=40)
>  [0]	HashMap\$Node<K,V> (id=51)
▼  [1]	HashMap\$Node<K,V> (id=52)
>  key	"IMO9197832" (id=71)
>  value	Tanker (id=72)
>  [2]	HashMap\$Node<K,V> (id=53)







▼  navireEnAttente	HashMap<K,V> (id=39)
>  [0]	HashMap\$Node<K,V> (id=65)
>  key	"IMO9379715" (id=66)
>  value	Tanker (id=67)
▼  navirePartis	HashMap<K,V> (id=40)
>  [0]	HashMap\$Node<K,V> (id=51)
>  key	"IMO9241061" (id=60)
>  value	Croisiere (id=61)
▼  [1]	HashMap\$Node<K,V> (id=52)
>  key	"IMO9394076" (id=55)
>  value	Tanker (id=56)

Vous allez maintenant vous occuper des cargos. et faire arriver 4 cargos. Il y a suffisamment de quais disponibles pour les accueillir.

Dans la méthode main :

```
/**
 * Dans ce test, on enregistre l'arrivée de 4 cargos
 * et il y a de la place.
 */
Test.testEnregistrerArriveePrevue(port, new Cargo("IMO9755933", "MSC DIANA", "39.74224 N", "5.99304 E",
193489, 202036, 176000, "Matériel industriel"));
Test.testEnregistrerArriveePrevue(port, new Cargo("IMO9204506", "HOLANDIA", "41.74844 N", "6.87008 E",
8737, 9113, 7500, "marchandises diverses"));
Test.testEnregistrerArrivee(port, "IMO9780859");
Test.testEnregistrerArrivee(port, "IMO9250098");
Test.testEnregistrerArrivee(port, "IMO9502910");
Test.testEnregistrerArrivee(port, "IMO9755933");
```

Il y a bien les 2 super tankers et les 4 cargos arrivés :

▼  navireArrives	HashMap<K,V> (id=33)
▼  [0]	HashMap\$Node<K,V> (id=51)
>  key	"IMO9220952" (id=59)
>  value	Tanker (id=60)
▼  [1]	HashMap\$Node<K,V> (id=52)
>  key	"IMO9755933" (id=64)
>  value	Cargo (id=65)
▼  [2]	HashMap\$Node<K,V> (id=53)
>  key	"IMO9780859" (id=67)
>  value	Cargo (id=68)
▼  [3]	HashMap\$Node<K,V> (id=54)
>  key	"IMO9250098" (id=69)
>  value	Cargo (id=70)
▼  [4]	HashMap\$Node<K,V> (id=55)
>  key	"IMO9379715" (id=71)
>  value	Tanker (id=72)
▼  [5]	HashMap\$Node<K,V> (id=56)
>  key	"IMO9502910" (id=73)
>  value	Cargo (id=74)





Vous allez maintenant enregistrer l'arrivée d'un cargo qui sera mis en attente faute de portiques disponibles :

```
/**
 * Dans ce test on va enregistrer l'arrivée d'un cargo qui sera mis en attente
 */
Test.testEnregistrerArrivee(port, "IMO9204506");
```

Le résultat d'exécution, vous devez constater :

- ✓ Que le navire n'est plus dans la liste des navires attentus
- ✓ qu'il n'est pas dans la liste des navires arrivés
- ✓ qu'il est dans la liste des navires en attente.











▼  navireEnAttente	HashMap<K,V> (id=39)
▼  [0]	HashMap\$Node<K,V> (id=51)
>  key	"IMO9204506" (id=54)
>  value	Cargo (id=55)

Vous allez maintenant enregistrer le départ d'un super tanker, le cargo devrait rester en attente :

```
/**
 * Dans ce test on va enregistrer le départ d'un super tanker
 * Le cargo devrait rester en attente
 */
Test.testEnregistrerDepart(port, "IMO9220952");
```









Vous devriez voir le super tanker parti et le cargo toujours en attente :

▼  navireEnAttente	HashMap<K,V> (id=39)
▼  [0]	HashMap\$Node<K,V> (id=51)
>  key	"IMO9204506" (id=54)
>  value	Cargo (id=55)
▼  navirePartis	HashMap<K,V> (id=40)
▼  [0]	HashMap\$Node<K,V> (id=63)
>  key	"IMO9220952" (id=70)
>  value	Tanker (id=71)

Et pour finir, vous allez enregistrer le départ d'un cargo, le cargo en attente devrait permuter dans les navires arrivés :

```
/**
 * Dans ce test on va enregistrer le départ d'un cargo
 * Le cargo en attente devrait passer dans les navires arrivés
 */
Test.testEnregistrerDepart(port, "IMO9755933");
```

Il n'y a plus de navires en attente et le navire en attente est bien arrivé.

▼  navireArrives	HashMap<K,V> (id=33)
>  [0]	HashMap\$Node<K,V> (id=53)
▼  [1]	HashMap\$Node<K,V> (id=54)
> ▲ key	"IMO9204506" (id=65)
> ▲ value	Cargo (id=66)
>  [2]	HashMap\$Node<K,V> (id=55)
>  [3]	HashMap\$Node<K,V> (id=56)
>  [4]	HashMap\$Node<K,V> (id=57)
>  navireAttendus	HashMap<K,V> (id=38)
 navireEnAttente	HashMap<K,V> (id=39)

A la fin du programme, vous devriez obtenir ceci :

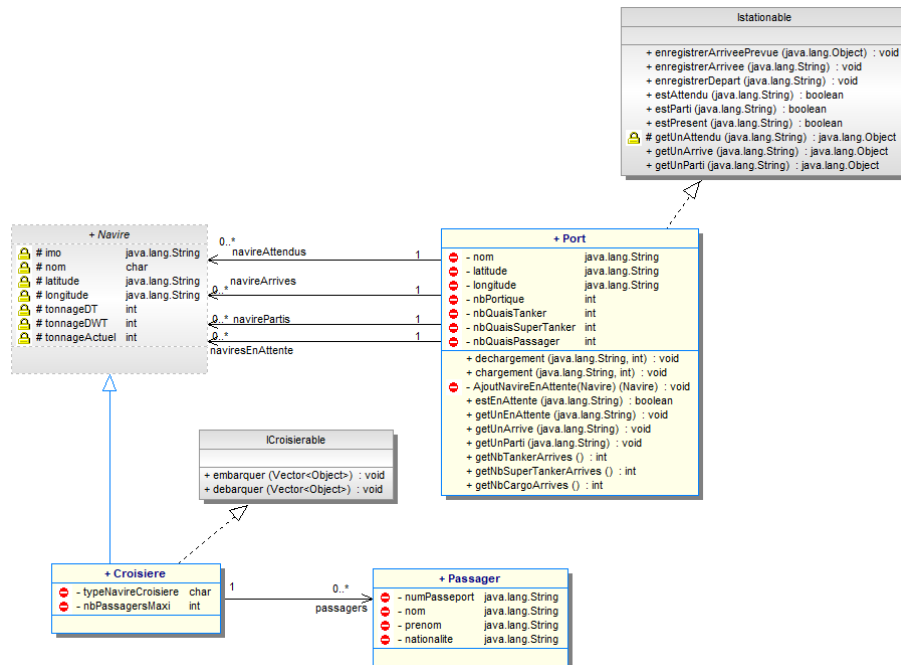
```
-----
System.out.println(port);
-----
```

```
Port de Marseille
Coordonnées GPS : 43.2976N / 5.3471E
Nb portiques : 4
Nb quais croisière : 4
Nb quais tankers : 3
Nb quais super tankers : 2
Nb Navires à quai : 5
Nb Navires attendus : 3
Nb Navires à partis : 5
Nb Navires en attente : 0
```

```
Nombre de cargos dans le port : 4
Nombre de tankers dans le port : 0
Nmbre de super tankers dans le port : 1
-----
```

## Partie 4 : GESTION DES PASSAGERS CROISIERISTES

En vous basant sur ce qui précède, il vous est demandé de gérer l'embarquement et le débarquement des passagers.



## Partie 5 : GESTION D'UN AEROPORT

Nouveau projet et réutilisation de l'interface IStationable et des classes :

- ✓ Aéroport
- ✓ Avion

Vous gèrerez ici les avions attendus, arrivés et partis... comme dans l'interface

## Partie 6 : ANNEXE

### 1. Classes

Classe navire (abstraite)	
Package : fr.roche.navireheritage.classesmetier	
Attributs protégés	
imo (lecture seule)	Numéro IMO du bateau sous la forme IMO99999999.
nom (lecture seule)	Nom du navire

latitude (lecture/écriture)	Position mise à jour à intervalles réguliers par un programme externe
longitude (lecture/ écriture)	Position mise à jour à intervalles réguliers par un programme externe
tonnageActuel	Partie du tonnage du bateau actuellement utilisée. Il est exprimé en tonnes
tonnageGT	Tonnage du bateau( <a href="https://fr.wikipedia.org/wiki/Tonnage">https://fr.wikipedia.org/wiki/Tonnage</a> )
tonnageDWT	Chargement maximal que peut embarquer un navire. Il comprend le personnel, les consommables (nourriture, fluides,...) et la cargaison. ( <a href="https://fr.wikipedia.org/wiki/Port_en_lourd">https://fr.wikipedia.org/wiki/Port en lourd</a> )
<b>Méthodes</b>	
+ Navire(String, String, String, String, int, int, int)	Constructeur permettant d'initialiser respectivement les attributs imo, latitude, longitude, nom, tonnageActuel, TonnageDT, tonnageDWT
+ getImo()	Accesseur
+ getLatitude()	Accesseur
+ setLatitude(String)	Mutateur
+ getLongitude()	Accesseur
+ setLongitude(String)	Mutateur
+ getNom()	Accesseur
+ getTonnageGT()	Accesseur
+ getTonnageDWT()	Accesseur
+ getTonnageActuel()	Accesseur
+ setTonnageActuel(int)	Mutateur

Classe cargo hérite de navire	
Package : fr.roche.navireheritage.classesmetier	
Attributs privés	
typeFret	Chaine représentant le type de cargaison du bateau : denrées périssables, matériel, ...
Méthodes	
Cargo(String, String, String, String, int, int, int, String)	Constructeur permettant de valoriser les attributs de la classe mère et le type de fret
charger(int)	Méthode qui met à jour le tonnage actuel du bateau avec la valeur passée en paramètre. La quantité passée en paramètre est enlevée à la quantité actuelle
decharger(int)	Méthode qui met à jour le tonnage actuel du bateau avec la valeur passée en paramètre. La quantité passée en paramètre est ajoutée à la quantité actuelle
getTypeFret()	accesseur

Classe Tanker hérite de navire	
Package : fr.roche.navireheritage.classesmetier	
Attributs privés	
typeFluide : chaine	Gaz liquide, pétrole, huile...
Méthodes	
+Tanker(String, String, String, String, int, int, int, String)	Constructeur permettant de valoriser les attributs de la classe mère et le type de fluide à bord du navire
+charger(int)	Méthode qui met à jour le tonnage actuel du navire avec la valeur passée en paramètre. La quantité passée en paramètre est enlevée à la quantité actuelle



+decharger(int)	Méthode qui met à jour le tonnage actuel du navire avec la valeur passée en paramètre. La quantité passée en paramètre est ajoutée à la quantité actuelle
+getTypeFluide()	accesseur

Classe Croisiere hérite de navire	
Package : fr.roche.navireheritage.classesmetier	
Attributs privés	
typeNavireCroisiere	V: voilier ; M: moteur
nbPassagersMaxi	Nombre de passager maximum que peut embarquer un navire
Passagers : HashMap<String, Passenger>	Dictionnaire d'objets de la classe Passenger identifiés par leur numéro de passeport. Représente les personnes croisiéristes à bord du bateau.
Méthodes	
Croisiere(String, String, String, String, int, int, int, char, int)	Constructeur permettant de valoriser les attributs de la classe mère, le type de navire de croisière et le nombre de passagers maximum.
Croisiere(String, String, String, String, int, int, int, char, int, Vector<Passager>)	Constructeur surcharge permettant en plus de charger une liste de passagers.
embarquer(Vector<Object>)	Méthode qui met à jour la liste des passagers du bateau. Les passagers passés en paramètres doivent être ajoutés de la liste des passagers du navire. Une exception sera générée si le nombre de passagers dépasse le nombre de passagers maximum du navire. Aucun passager ne sera alors ajouté.
debarquer(Vector<Object>)	Méthode qui met à jour la liste des passagers du bateau. Les passagers passés en paramètres doivent être retirés de la liste des passagers du navire. Cette méthode retourne la liste des passagers passés en paramètre et qui n'ont pas été trouvés dans la liste des passagers du navire.
getTypeFret()	accesseur
getNbPassagersMaxi()	
getPassagers()	
getTypeNavireCroisiere()	

Classe Passenger	
Package : fr.roche.navireheritage.classesmetier	
Attributs privés	
numPasseport	Chaine de caractères
nom	
prenom	
nationalite	
Méthodes	
+ Passenger(String, String, String, String)	Constructeur permettant de valoriser les attributs de la classe.
+ getNumPasseport()	accesseur
+ getNom()	accesseur
+ getPrenom()	accesseur
+ getNationalite()	accesseur

Classe Port	
Package : fr.roche.navireheritage.classesmetier	
Attributs privés	
Nom : String	du port
Latitude : String	du port
Longitude : String	du port
nbPortique: int	Nombre de points d'accueil d'un cargo
nbQuaisPassager: int	Nombre de quais d'accueil pour navires passagers
nbQuaisTanker: int	Nombre de quais d'accueil pour les tankers de jusqu'à 130000 tonnes (GT)
nbQuaisSuperTanker: int	Nombre de quais d'accueil pour les tankers de plus de 130000 tonnes(GT)
navireAttendus ; HashMap<String, Navire>	Dictionnaire des navires attendus. String = id du navire
navireArrives: HashMap<String, Navire>	Dictionnaire des navires arrivés, c'est-à-dire présents dans le port. String = id du navire
navirePartis : HashMap<String, Navire>	Dictionnaire des navires partis récemment. String = id du navire
navireEnAttente : HashMap<String, Navire>	Dictionnaire des navires en attente d'avoir un quai libre pour stationner. String = id du navire
Méthodes	
+ Port(String, String, String, int, int, int, int)	
+ enregistrerArriveePrevue(Object)	Enregistrement de l'arrivée proche d'un navire
+ enregistrerArrivee(Object)	Méthode surcharge : enregistrement de l'arrivée d'un navire enregistré en tant que arrivée prévue. String = son id
+ enregistrerDepart(Object)	enregistrement du départ d'un navire présent dans le port. String = son id On ne peut enregistrer un départ que si le Navire est présent dans le port.
- AjoutNavireEnAttente(Object)	Ajout du navire passé en paramètre dans le dictionnaire des navires en attente d'un quai dans le port.
+ estAttendu(String)	Retourne vrai si le navire est attendu. Id= imo
+ estPresent(String)	Retourne vrai si le navire est dans le port. Id= imo
+ estEnAttente(String)	Retourne vrai si le navire est en attente d'un quai dans le port. Id= imo
+ chargement(String, int)	Chargement du navire dont l'id est passé en paramètre de la quantité passée en paramètre String : Id du navire Qté : quantité à décharger
+ dechargement(String, int)	Déchargement du navire dont l'id est passé en paramètre de la quantité passée en paramètre String : Id du navire Qté : quantité à décharger
+ getNom() : String	accesseur
+ setNom(String) : int	mutateur
+ getLatitude(): String	accesseur
+ getLongitude(): String	accesseur
+ getNbPortique(): int	accesseur
+ setNbPortique(int) : int	mutateur
+ getNbQuaisPassager() : int	accesseur

+ setNbQuaisPassager(int)	mutateur
+ getNbQuaisTanker(): int	accesseur
+ setNbQuaisTanker(int)	mutateur
+ getNbQuaisSuperTanker(): int	accesseur
+ setNbQuaisSuperTanker(int) : int	mutateur
+ getNavireAttendus() : HashMap<String, Navire>	accesseur
+ getNavireArrives() : HashMap<String, Navire>	accesseur
+ getNavirePartis() : HashMap<String, Navire>	Accesneur
+ getNavireEnAttente() : HashMap<String, Navire>	Accesneur
getUnAttendu(String id): Object	Retourne l'objet dont l'id a été passé en paramètre dans le dictionnaire ou une exception de type Exception
getUnArrive(String id): Object	Retourne l'objet dont l'id a été passé en paramètre ou une exception de type Exception
getUnEnAttente(String id): Object	Retourne l'objet dont l'id a été passé en paramètre ou une exception de type Exception
getUnParti(String id): Object	Retourne l'objet dont l'id a été passé en paramètre ou une exception de type Exception
getNbTankerArrives() : int	retourne le nombre de tankers (tonnageGT <= 130000) présents dans le port
getNbSuperTankerArrives()	Retourne le nombre de super tankers (tonnageGT>130000) présents dans le port
getNbCargoArrives()	retourne le nombre de cargos présents dans le port

Classe Utilitaire	
Package : fr.roche.navireheritage.classestechnique	
Attributs privés	
Méthodes	
itineraireDeChargeNavire(Navire, int) : HashMap<Integer,Integer>	Va rechercher dans le port les stockages à parcourir pour décharger le navire passé en paramètre de la quantité passée en paramètre.
itineraireChargeNavire(Navire, int) : HashMap<Integer,Integer>	Va rechercher dans le port les stockages à parcourir pour charger le navire passé en paramètre de la quantité passée en paramètre.

Classe Stockage	
Package : fr.roche.navireheritage.classesmetier	
Attributs privés	
numero : int	Identifiant du stockage. Lecture seule
capaciteMaxi : int	Capacité de tonnage maximum du stockage
capaciteDispo : int	Capacité disponible du stockage
Méthodes	
Stockage(int, int)	Constructeur permettant de valoriser les attributs numero et capaciteMaxi . La capaciteDispo sera valorisée à capaciteMaxi
getNumero()	accesseur
getCapaciteMaxi() : int	accesseur
setCapaciteMaxi(int)	mutateur

getCapaciteDispo() : int	accesseur
setCapaciteDispo(int)	mutateur

## 2. interfaces

Interface iNavCommercable	
Package : fr.roche.navireheritage.interfaces	
Cette interface va imposer les méthodes nécessaires à la gestion de tout navire marchand. Dans notre exemple il s'agira du chargement et déchargement.	
Méthodes	
decharger(int qte)	Méthode qui met à jour le tonnage actuel du navire avec la valeur passée en paramètre. La quantité passée en paramètre est enlevée à la quantité actuelle.
charger(int qte)	Méthode qui met à jour le tonnage actuel du bateau avec la valeur passée en paramètre. La quantité passée en paramètre est ajoutée à la quantité actuelle.

Interface ICroisierable	
Package : fr.roche.navireheritage.interfaces	
Cette interface va imposer les méthodes nécessaires à la gestion de tout navire de croisière. Dans notre exemple il s'agira de l'embarquement et du débarquement de passagers afin de maintenir à jour la liste des passagers croisiéristes du navire.	
Méthodes	
embarquer(Vector<Object> objects)	Méthode qui met à jour le tonnage actuel du navire avec la valeur passée en paramètre. La quantité passée en paramètre est enlevée à la quantité actuelle.
debarquer(Vector<Object> objects)	Méthode qui met à jour le tonnage actuel du bateau avec la valeur passée en paramètre. La quantité passée en paramètre est ajoutée à la quantité actuelle.

Interface Istationable	
Package : fr.roche.navireheritage.interfaces	
Cette interface va imposer les méthodes nécessaires à la gestion de toute infrastructure gérant des objets de passage. Ici il s'agit de Port qui gère les arrivées et départs de navires, mais il pourrait s'agir de trains ou d'avion.	
Méthodes	
enregistrerArriveePrevue(Object)	Méthode qui met à jour la liste des objets qui sont susceptibles d'arriver dans la station (port, aéroport,...)
enregistrerArrivee(String)	Méthode qui enregistre l'arrivée réelle de l'objet.
enregistrerDepart(String)	Méthode qui enregistre le départ d'un objet présent dans la station.
estAttendu(String id) : booléen	Retourne vrai si l'objet dont l'id est passé en paramètre fait partie des objets attendus dans la station
estPresent(String id) : booléen	Retourne vrai si l'objet dont l'id est passé en paramètre fait partie des objets présents dans la station
estParti(String id) : booléen	Retourne vrai si l'objet dont l'id est passé en paramètre est parti de la station depuis peu de temps
getUnAttendu(String id): Object	Retourne l'objet dont l'id a été passé en paramètre ou une exception de type Exception

getUnArrive(String id): Object	Retourne l'objet dont l'id a été passé en paramètre ou une exception de type Exception
getUnParti(String id): Object	Retourne l'objet dont l'id a été passé en paramètre ou une exception de type Exception

*Fin du TD .....*

