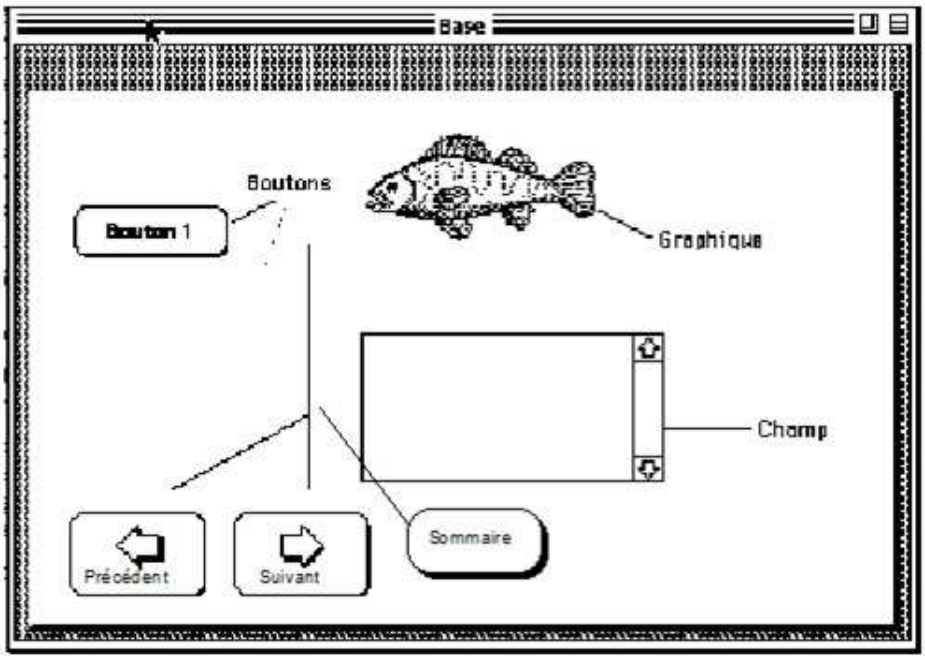
Exercice 25 - UML - Diagramme de classes - Diagrammes de séquence

Hypercard est une application apparue dans les années 1980 et qui était très en avance sur son époque. En effet, elle contenait déjà les concepts objet ainsi que la navigation hypertexte.
Les documents manipulés par Hypercard sont appelés des "**piles**". Une pile est composée de "**cartes**" (à l'image des fiches bristol, ancêtres en papier du concept de fichier). Toute pile comporte une carte particulière appelée la **carte sommaire** qui reste toujours sur le dessus de la pile. L'ordre des autres cartes peut être changé mais la notion d'ordre des cartes est toujours conservée : toute carte a une suivante (sauf évidemment la dernière), et une précédente (sauf évidemment la carte sommaire qui est la première). La carte sommaire est utilisée généralement pour faire une "hyper table des matières" permettant d'accéder directement à des cartes particulières appelées **cartes "tête de chapitre"**.
Les cartes contiennent des "**widgets**" (window objects) qui peuvent être des **boutons**, des **champs texte** (pouvant servir pour saisir une entrée ou afficher un résultat) et des **graphiques bitmap** ou **vectoriel**. Un widget qui doit figurer dans toutes les cartes est placé sur un "**fond de carte**" (équivalent à ce que l'on appelle "masque de diapositive" dans l'application Powerpoint).
Les widgets possèdent un script (pouvant faire des tâches relativement complexes comme des calculs) écrit dans le langage Hypertalk (petit langage objet). Les méthodes du script sont exécutées lorsque l'objet widget reçoit un message. Par exemple, le messages `onClick()` est envoyé à l'objet lorsqu'un utilisateur clique sur l'objet. D'autres messages provenant de l'utilisateur sont par exemple envoyés à un champ texte lorsqu'une donnée y est saisie, etc.
On ne s'intéressera dans cet exercice qu'aux boutons qui permettent la navigation hypertexte : lorsque l'on clique sur un de ces boutons, le script exécuté permet de naviguer jusqu'à une autre carte. Ces boutons particuliers sont appelés **"boutons lien"**. Il existe trois boutons liens particuliers (prédéfinis dans une bibliothèque de Hypercard) : les boutons "précédent", "suivant" et "sommaire" qui permettent respectivement la navigation vers la carte précédente, la carte suivante suivante et la carte sommaire (voir Figure). Il existe aussi un autre type de bouton lien appelé "bouton chapitre". Ces boutons sont généralement utilisés dans la carte sommaire pour offrir un accès rapide aux cartes tête de chapitre.



Dans l'exercice, on fera les hypothèses suivantes :

- L'affichage d'une carte est réalisé en lui envoyant un message `afficher()`. La carte s'adresse alors à tous ses widgets en leur envoyant un message afficher().
- Tout bouton possède un attribut nommé `intitulé` qui contient une chaine de caractère (éventuellement vide). Les boutons "précédent", "suivant" et "sommaire" sont identifiés par cet attribut contenant ces mêmes chaînes de caractères. D'autres attributs peuvent être ajoutés : en particulier les boutons chapitres possèdent un attribut supplémentaire appelé `numéro` donnant le numéro du chapitre.
- La responsabilité de la navigation est située au niveau des cartes. Chaque carte connait sa carte précédente et sa carte suivante et la carte sommaire connait toutes les cartes tête de chapitre. Les boutons sont de simples objets réactifs dont la seule fonction est d'informer la carte à laquelle ils appartiennent qu'ils viennent d'être cliqués. De plus la carte ne connaît alors que l'identité du bouton, et doit se débrouiller pour récupérer les attributs dont elle a besoin.

Construire un modèle UML cohérent de l'univers décrit dans le texte précédent.
Ce modèle sera composé d'un diagramme de classes.
Il sera aussi composé de diagrammes de séquences illustrant des scenarios significatifs de navigation. Les scénarios suivant seront modélisés :

- L'utilisateur exécute l'application qui affiche la carte sommaire. Il clique sur un bouton chapitre de la carte sommaire. Après avoir lu la carte correspondante, l'utilisateur retourne sur la carte sommaire.

- L'utilisateur exécute l'application qui affiche la carte sommaire. Il clique sur le bouton suivant de la carte puis re-clique sur le bouton suivant de la carte sur laquelle il est arrivé.

Exercice 26 - UML - Diagramme de classes - Diagrammes de séquence

Dans l'application ProjectCalendar, on suppose qu'il existe un module graphique `TacheFilter` qui permet d'intéragir avec l'utilisateur de l'application afin de rechercher des taches correspondantes à différents critères.

L'utilisateur peut entrer l'identificateur ou le début de l'identificateur des taches recherchées dans un champs texte. L'utilisateur peut choisir de filtrer les tâches d'un projet particulier en le choisissant dans une liste déroulante (un objet "ComboBox"). L'utilisateur peut filtrer les tâches au fait qu'elles soient préemptives ou non, et qu'elles aient déjà été programmées ou non. Pour cela, l'utilisateur coche les "CheckBoxes" des deux groupes `préemptives, non-préemptives` et `programmées, non-programmées` du module graphique pour préciser les tâches concernées par la recherche.

Dans cet exercice, il s'agit de construire un diagramme de séquence modélisant le scénario ci-dessus tout en complétant un diagramme de classes de façon à le rendre cohérent par rapport à ce diagramme de séquence. On fera les hypothèse suivantes :

- La méthode `onEdit()` du `TacheFilter` est appelée si il y a eu une modification sur l'un des éléments d'interaction de l'objet `TacheFilter`.
- Les méthodes `isPreemptiveChecked()`, `isNotPreemptiveChecked()`, `isScheduledChecked()`, `isNotScheduledChecked()` de `TacheFilter`, qui renvoient toutes une valeur de type **`bool`**, permettent de savoir quelles sont les cases cochées parmi les composants de type "CheckBox".
- La méthode `getId()` de `TacheFilter` permet de récupérer (dans un `QString`) l'identificateur (ou le début de l'identificateur) entré dans la barre d'édition de l'objet `TacheFilter`.
- La méthode `getProject` de `TacheFilter` permet de récupérer (dans un `QString`) le nom du projet sélectionné dans la liste déroulante du `TacheFilter`.
- La méthode `addTache(t:Tache&)` de `TacheFilter` permet d'ajouter identificateur de tâche dans la partie du `TacheFilter` destinée à afficher le résultat du filtre de la recherche.
- La méthode `clear` de `TacheFilter` permet de réinitialiser la partie de l'objet `TacheFilter` destinée à afficher le résultat du filtre de la recherche.
- La classe `TacheManager` possède une classe `IdIterator` qui permet d'itérer sur toutes les taches dont l'identificateur commence par un texte donné. Pour cela la méthode `getIdIterator(i:QString)` de `TacheManager` permet d'obtenir un itérateur qui pointe sur la première tâche dont l'identificateur commence par la chaîne `i`. Les méthodes `currentItem()`, `next()` et `isDone()` permettent respectivement d'obtenir une référence sur la tâche courante, de se déplacer sur la prochaine tâche, et de savoir si l'itérateur est arrivé à la fin de la séquence d'itération.