TD2 : Les hamsters ... la suite

Ce TD fait suite au TD1. Dans le TD1, nous avons vu comment développer une application selon le patron de conception "Modèle - Vue - Contrôleur" (MVC).

- Dans ce TD, nous complétons le programme entamé au TD1 pour obtenir une interface fonctionnelle. Vous pouvez vous inspirer de la correction proposée

Interface graphique

Nous allons programmer une interface graphique avec la librairie appJar (voir aussi S5-POO)

L'écriture d'une interface graphique permet de mettre en œuvre les principes de la *programmation événementielle*:

- Une interface est constituée de plusieurs éléments graphiques (appelés *widgets*) positionnés dans une fenêtre.
  - les widgets sont éditables/positionnables via les méthodes définies dans la librairie
  - il est possible de modifier leur contenu *en cours d'exécution* (via des *événements*)
- Les actions de l'utilisateur (mouvements et clics de souris, entrées clavier ,...) produisent des *événements*:
  - Les événements sont déclenchés par des widgets actifs (boutons, menus,...) capables de lancer l'exécution d'une ou plusieurs opérations:
    - mise à jour d'une variable ou d'un attribut
    - lancement d'un calcul
    - envoi d'un message
    - modification du rendu visuel
    - arrêt du programme
    - etc.
- Chaque élément graphique est caractérisé par son identifiant unique
  - créé au moment de son initialisation
  - utilisable au sein de l'espace des noms de l'application :
    - (donc par toute fonction qui reçoit en paramètre le descripteur de l'application)
  - permettant de changer le rendu graphique en cours d'exécution.

Dans le cadre du patron MVC, les informations affichées dans l'interface (le contenu des widgets) sont paramétrées par les objets du modèle. On a typiquement le cycle suivant :

- L'utilisateur déclenche une action via l'interface
- Le contrôleur exécute l'action
- Les changements produits par l'action sont répercutés dans le modèle
- l'interface consulte le modèle pour mettre à jour son contenu.

A faire

Ouvrez Pycharm et reprenez le projet du TD1.

Pour rappel, le programme sert à gérer une petite animalerie (constituée de rongeurs divers...). Les animaux sont décrits dans le fichier `animal.json` et les équipements (litière, roue, mangeoire,...)

dans le fichier `equipement.json`.

- le modèle définit les opérations permettant la mise à jour de l'état des animaux et de l'occupation des équipements
- le contrôleur définit les actions possibles, à savoir nourrir, divertir, coucher et réveiller les animaux. Les actions sont valides (ou non) en fonction de l'état des animaux et de l'occupation des équipements. Des messages de mise en garde s'affichent lorsque l'action choisie n'est pas valide.

- Exécutez les tests pour vérifier que tout fonctionne bien

Construction de l'interface

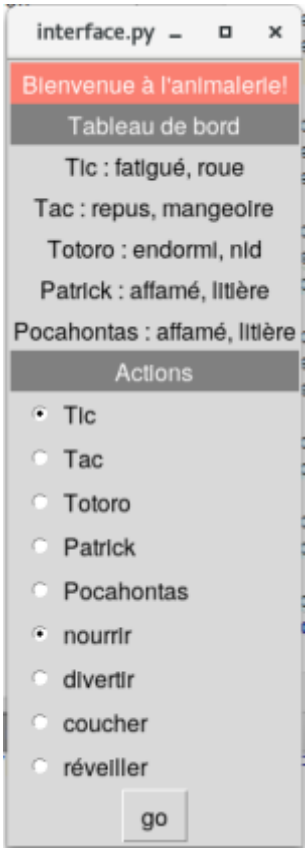
- Ajoutez un module `vue.py`
- Pour programmer l'interface nous utilisons la librairie appJar:

```
from appJar import gui
```

- On initialise une application (correspondant à une fenêtre graphique)

```
app = gui()
```

Le but est de construire une interface très simple:



Le premier bandeau (rose saumon) peut être défini à l'aide des commandes suivantes :

```
app.addLabel("en-tête", "Bienvenue à l'animalerie!")
app.setLabelBg("en-tête", "salmon")
```

```
app.setLabelFg("en-tête", "white")
```

- Un widget de type `Label` est ajouté à l'application.
- Son *index* (autrement dit son identifiant) est `"en-tête"`
- Pour modifier son rendu, on applique un setter : `app.setXXX…` indexé par `"en-tête"`.

Pour tester le rendu, il suffit d'ajouter la commande:

```
app.go()
```

et d'exécuter `vue.py`.



**A FAIRE: **

Définissez un deuxième bandeau de couleur grise (`"gray"`) affichant le texte `"Tableau de bord"` en blanc, et exécutez la vue pour vérifier l'affichage



Pensez à définir un nouvel index

Pour afficher l'état des animaux, il faut à présent consulter le modèle.

```
import modele
```

- La liste des animaux n'étant pas donnée par le modèle, nous la définissons dans la vue :

```
liste_animaux = ['Tic', 'Tac', 'Totoro', 'Patrick', 'Pocahontas']
```



** A FAIRE: **

Pour chaque animal de la liste :

- Utiliser le modèle pour connaître l'état de l'animal ainsi que son lieu
- Initialiser un widget de type Label,
  - dont l'index est le nom de l'animal
  - et dont le contenu est un texte indiquant le nom de l'animal, son lieu et son état
- Exécuter la vue pour vérifier que les animaux s'affichent bien.

La deuxième partie de la vue est constituée de deux listes à choix multiples pour choisir l'action à effectuer :

- choix de l'animal
- et choix de l'action

Le bouton Go permet de lancer l'exécution, *via* un appel au contrôleur.

Listes à choix multiples

Dans l'exemple présenté, les listes à choix multiples sont réalisées par des widgets de type "RadioButton" permettant de fixer une valeur. Nous avons deux valeurs à définir :

- l'identifiant de l'animal sur lequel agir
- le choix de l'action



- Les boutons radio servant à gérer une même variable ont le même index
- chaque bouton sert à instancier une valeur différente
- la valeur courante est obtenue en appelant la méthode `getRadioButton(index)`

**A FAIRE: **

- pour chaque animal a de la liste des animaux, initialiser un bouton radio indexé par `"id_animal"`:

```
app.addRadioButton("id_animal", a)
```



- définissez la liste des actions possibles
- pour chaque action c de la liste des actions initialisez un bouton radio indexé par `"action"`:

```
app.addRadioButton("action", c)
```

- exécutez la vue et vérifiez que les boutons radio permettent bien la sélection de valeurs.

Boutons d'action

On ajoute à présent le bouton d'action qui permet d'exécuter les actions définies dans le contrôleur:

On commence par importer le contrôleur:

```
import controleur
```



Rappel : le contrôleur propose les actions : nourrir, divertir, endormir et réveiller

Exemple :

```
controleur.nourrir(id_animal)
```

a pour effet de modifier *dans le modèle* l'état de l'animal choisi :



- il est déplacé vers la mangeoire (si elle n'est pas occupée)
- et l'animal passe de l'état affamé à repus

Dans l'interface, un bouton (Button) permet de lancer l'exécution d'une fonction selon la syntaxe:

```
app.addButton("go",  press)
```

**A FAIRE: **

Définir la fonction :

```
def press(act):
    ...
```

qui exécute une action du contrôleur selon les deux valeurs :

- `app.getRadioButton("action")` : le nom de l'action
- `app.getRadioButton("id_animal")` : l'identifiant de l'animal



ainsi si l'action vaut `"nourrir"` alors il faut exécuter :

```
controleur.nourrir(app.getRadioButton("id_animal"))
```

etc.

- Codez les appels aux actions nourrir, divertir, coucher et réveiller dans la fonction
- Exécutez la vue
- Pour vérifier que les actions sont bien prises en compte, effectuez une action *invalide*. Si l'action n'est pas valide, un message du type `Attention XXX n'a pas faim` doit en effet s'afficher dans la console.

Ce n'est pas fini !

Lorsqu'elles sont valides, les actions modifient le contenu du modèle, mais ces changements ne sont pas visibles au niveau de l'interface. Pour répercuter le résultat de l'action dans la vue, il faut donc mettre à jour le widgets correspondant à l'affichage de l'état des animaux (le "tableau de bord").



Les identifiants d'animaux servant aussi d'index pour les widgets du tableau de bord, on modifie le contenu d'un widget à l'aide de :

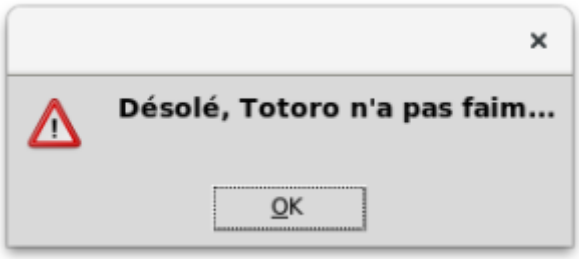
```
app.setLabel(id_animal, ...)
```

**A FAIRE: **

- Écrire une fonction qui met à jour l'ensemble des widgets du tableau de bord en consultant le modèle à partir de la liste des animaux (comme dans la partie initialisation)
- Ajouter un appel à cette fonction à la fin de la fonction `press()`
- Exécuter la vue et vérifier que le tableau de bord est bien modifié lorsque les actions sont valides.

Améliorations

Il est préférable lorsque l'action n'est pas valide d'afficher le message d'erreur dans une fenêtre *pop-up* du type :



Les fenêtre à message sont exécutées à l'aide d'une commande:

```
app.warningBox("", texte)
```

** A FAIRE: **

- Comme les messages d'erreur sont générés dans le contrôleur, il faut modifier les fonctions nourrir, divertir, coucher et réveiller pour qu'elles retournent un texte qui sera ensuite affiché dans une fenêtre popup par la vue...
- Définir également un message lorsque l'action est valide (exemple `"Félicitations, Totoro a rejoint le nid et est maintenant endormi."`)
- Les actions valides apparaissent dans des `infoBox` et les actions invalides dans des `warningBox`

Interface tabulaire

Il est possible d'améliorer le rendu visuel de la vue en utilisant un positionnement tabulaire pour les widgets. Vous êtes invités à consulter la documentation pour obtenir un rendu du type:



NB :

- pour le bleu pâle : `app.setLabelBg(label, "lavender")`
- pour aligner à gauche : `app.setLabelAlign(label, "left")`

From:
https://wiki.centrale-med.fr/informatique/ - **WiKi informatique**

Permanent link:
https://wiki.centrale-med.fr/informatique/public:appro-s7:td2

Last update: **2023/10/15 22:37**