|  |  |
| --- | --- |
| Exercices-ateliers | **Les threads et l’animation** |

1. Chargez le projet *AnimationControlee.* Pour l’instant, le cercle jaune n'est pas encore visible, aucune animation n'est en cours, et les boutons ne fonctionnent pas.
   1. On veut que l’animation démarre au clic du bouton *Go*. Pour ce faire, appelez la méthode *demarrer* offerte pour la classe *ZoneAnimation*. Le cercle jaune se déplacera alors de 2.3 pixels à la fois. Le bouton *Stop* ne fonctionne toujours pas.
   2. Cliquez à répétition sur le bouton *Go*. Comment expliquez-vous le résultat observé? Blindez votre application contre ce résultat indésirable.
      * Vous devez modifier la méthode *demarrer* pour éviter qu’un deuxième thread ne soit créé si l’animation est déjà en cours. Pour ce faire, déclarez un champ booléen *enCoursDAnimation*, et gérez-le adéquatement dans la méthode *demarrer*.
   3. Ajoutez à l'animation une pizza volante*.* La pizza doit apparaitre progressivement en provenance de la droite, et se déplacer vers la gauche.
      * Il vous faudra utiliser une variable, disons *xPizz*, pour suivre la position de la pizza, et l'initialiser en utilisant la largeur actuelle du composant (rappel : cette largeur n'est pas connue dans le constructeur ni dans les déclarations des champs)
      * La méthode *drawImage* exige des positions entières: vous devrez utiliser *(int)xPizz* pour cette méthode (mais il est toutefois important que *xPizz* soit un *double*, pour accumuler convenablement toutes les fractions de pixels du déplacement cumulatif!!).
      * La pizza doit être sensiblement la même taille que le cercle, et être à la même hauteur (utilisez POS\_Y et DIAMETRE)
      * Quand elle croise le cercle jaune, le cercle doit être à l'avant-plan
   4. Le bouton *Stop* doit maintenant fonctionner. Pour arrêter l'animation, vous devrez faire mourir le thread, c'est-à-dire mettre fin à la méthode *run* (rappel: la méthode *stop* de la classe *Thread* ne doit jamais être utilisée!!). Suivez les étapes ci-dessous:
      * En ce moment, *run* possède une boucle *while* infinie. Modifiez-la ainsi:

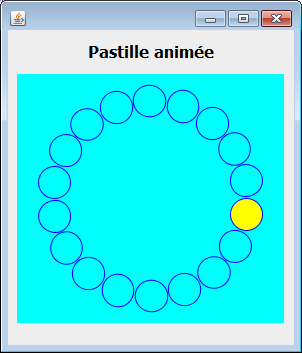
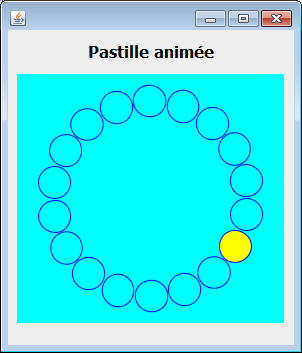
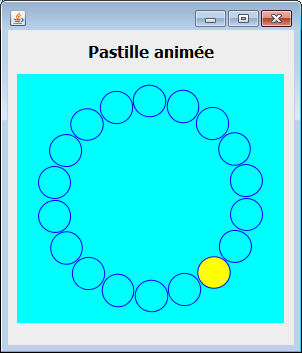
while (*enCoursDAnimation*) {

* Complétez ensuite la méthode *arreter* de la classe ZoneAnimation. Dans la classe Go, appelez cette méthode au clic du bouton *Stop*.
* Testez que les boutons Stop et Go peuvent désormais être utilisés, dans un ordre quelconque.
  1. Ajoutez à la classe *ZoneAnimation* un écouteur de souris: quand l'utilisateur clique sur le composant, l'animation change d'état (démarre ou s'arrête, selon le cas). Assurez-vous que ce comportement est encapsulé, c'est-à-dire que toute instance de type *ZoneAnimation* se comporte ainsi.
  2. Assurez-vous que vous pourriez répondre aux questions théoriques suivante :
  + ❓ Décrire deux approches différentes qui pourraient être utilisées pour que le cercle et la pizza se déplacent plus "vite" qu'ils ne le font présentement (mais sans toutefois accélérer).
  + ❓Serait-il possible de réaliser les déplacements du cercle et de la pizza en utilisant des transformations affines? Si oui, comment feriez-vous?
  + ❓Seriez-vous capable de tracer le diagramme UML de ce projet?
  1. ✪ Pour les pros : Le cercle doit constamment "clignoter" entre les couleurs jaune et bleu. Chaque couleur doit être visible environ 1 seconde avant de passer à la suivante. Dans cette application, quelles sont les trois variables d'animation dont *run* sera responsable?

1. Chargez le projet *PastilleAnimee.* Pour l’instant cette application affiche une série de pastilles disposées en cercle.

Vous devez la modifier pour qu’elle affiche également une pastille jaune (entourée de bleu) qui s’anime en tournant en cercle.

* + - Pour ce faire, le composant *Pastilles* doit être converti en composant animé. Celui-ci doit démarrer dès le démarrage de l'application : faites l’appel dans la classe *Application*.
    - Interrogez-vous: quelle sera la (ou les) variable d'animation? Assurez-vous que c'est la méthode *run* qui s'occupe de la (les) faire progresser.
    - Faites une pause fixe de 1 seconde entre chaque image de l'animation.

*Voici les 3 premières images qui seront affichées par l’application*

1. Chargez le projet *Mascottes*. Celui-ci contient une application destinée à afficher trois instances d'une petite mascotte animée. L'animation se fera en affichant tour à tour chacune des 10 images déjà liées au projet. Les images sont nommées *T1.gif, T2.gif, …T10.gif.*
   1. ❓Dans la classe *Mascotte*, pourquoi le tableau *images* est-il déclaré *static* ?
   2. Mettez l'arrière-plan de l'application en blanc. Ensuite, modifiez le composant *Mascotte* de manière à réaliser l'animation cyclique demandée. Quelle sera la (ou les) variable d'animation? Faites-la (les) progresser dans la méthode *run*.
   3. ✪ Pour les pros : Trouvez un moyen de faire en sorte que les 3 animations soient décalées dans le temps (et non synchronisées comme elles le sont présentement).
2. ✪ Pour les pros : Chargez le projet *Horloge*. Celui-ci contient une instance d'un composant graphique qui représente une horloge. Examinez les méthodes à votre disposition dans la classe *Horloge*. On aimerait que l'horloge soit mise à jour à toutes les secondes. Pour ce faire:

* Ajoutez l'unique instruction nécessaire, dans la classe *Application*, pour que le thread de la classe Horloge démarre au moment où l'utilisateur appuie sur le bouton *Démarrer*.
* Modifiez la classe Horloge de manière à ce que l'affichage soit mis à jour à chaque seconde par le thread. La méthode *heureEnChaine* retourne une chaîne contenant l'heure actuelle, dans le format requis, où des zéros ont été ajoutés si besoin.
* ❓ Si cette horloge était ajoutée à une application complexe (plusieurs composants animés, géométrie complexe, images nombreuse, etc), serait-il possible que sa précision (en tant qu'horloge) en soit affectée?

1. ✪ Pour les pros : Avec trois cercles, créez une petite animation approximative de la lune (bleue) qui tourne autour de la terre (grise), qui elle, tourne autour du soleil (jaune). Le soleil, immobile, sera bien centré dans le composant. Ajoutez du texte graphique (*drawString*) avec le nom de chacun, qui suivra chacun des astres.

|  |
| --- |
| **À garder à portée de mains !**  Examinez le projet *StrutureDUneAnimation* qui vous a été distribué. Il contient le squelette d'une application générale avec composant animé, incluant des commentaires vous rappelant ce que chaque méthode devrait contenir. |