|  |  |
| --- | --- |
| Exercices formatifs | **Événements**  **de la souris et du clavier** |

1. Dans cet exercice, nous développerons une petite application de dessin par points. Chargez le projet *DessinPointilles*. L'application qui s'y trouve permet de tracer un point quelconque. Le curseur et le bouton *Effacer* ne fonctionnent pas pour l'instant.
   1. Modifier la classe *DessinPointilles* de manière à tracer tous les points quand la souris se déplace en mode cliquer-glisser.
      * Utilisez tous les champs qui sont déjà déclarés dans le projet.
      * N'hésitez pas à recréer le cercle à l'intérieur de votre boucle de dessin (dans cette application, le temps supplémentaire requis est à peu près le même que pour l'utilisation alternative des transformations).
      * Assurez-vous qu'aucun point n'est affiché au démarrage.
   2. Ajoutez ensuite le code nécessaire au fonctionnement du bouton "Effacer", qui efface la surface de dessin et permet de recommencer à neuf. Indice: vous devrez utiliser une méthode de la classe ArrayList.
   3. Ajoutez du texte graphique (*drawString*) au bas de la surface de dessin; la phrase "Trait en cours" doit s'y afficher seulement quand c'est le cas (voir demo). Vous pouvez utiliser:

*g2d.drawString("Trait en cours....", 10, getHeight() - 20);*

* 1. Permettez à l'utilisateur d'ajuster le diamètre des cercles à l'aide du curseur. L'effet doit être immédiat et s’appliquer à tous les cercles présents et à venir. Assurez-vous que les cercles sont bien centrés sur la pointe du curseur.
  2. Au survol du bouton "Effacer", le fond du bouton doit virer au rouge (et reprendre sa couleur jaune quand la souris est ailleurs).
  3. Test temporaire: vérifiez que si vous placez une deuxième instance du composant *DessinPointilles* sur votre application, on peut sans problème y dessiner, et que le texte "Trait en cours" s'y affiche également (sinon, vous avez mal encapsulé les comportements demandés!).
  4. ❓ Réflexion : comment pourriez-vous améliorer cette application pour qu’on ait l’impression que le dessin se fasse « en continu » plutôt qu’en points séparés par des espaces?
  5. ✪ Pour les pros : un double-clic (n’importe où sur la surface de dessin) doit permettre de changer la couleur de tous les cercles pour une couleur aléatoire.

1. Chargez le projet *AffichagesSuccessifs*. Il ne contient qu'un composant personnalisé *FormesColorees* pour l'instant. Examinez cette classe.
   1. Avec WindowBuilder, créez une classe graphique démarrable nommée *Application*. N'oubliez pas d’utiliser un *Layout* de type *Absolute*.
   2. Ajoutez à la Palette le composant personnalisé *FormesColorees*. Placez ensuite deux instances de *FormesColorees* sur l'interface *Application*.
   3. Voici le comportement attendu pour chacune des instances: on veut qu'au démarrage, aucune forme ne soit affichée. Puis lorsque l'utilisateur fait un clic sur le composant, seule la forme rouge s'affiche. Au prochain clic, la forme bleue s'ajoutera. Au suivant, la jaune. Au suivant, elles disparaissent toutes. Au suivant, la forme rouge apparait, puis la bleue, etc… Pour ce faire, vous devrez associer un écouteur de souris au composant *FormesColorees*.

Ouvrez ***FormesColorees.java*** avec clic-droit*/Open With/WB Editor*, et passez en mode *Design*. Avec un clic droit sur la zone noire (le composant!), ajoutez un écouteur de souris en redéfinissant *mouseClicked*. Dans cet écouteur, ajoutez les lignes de code suivantes :

nbClicks = nbClicks + 1;

System.out.println ("Nombre de clics=" + nbClicks);

…et déclarez ensuite ce champ au haut de la classe:

private int nbClicks = 0;

* Testez votre application. Ce code de départ vous aidera à comprendre comment le compteur est mis à jour.
* A vous ensuite de modifier le code de la classe *FormesColorees* en y ajoutant les instructions nécessaires pour que son comportement soit tel que demandé. Vous devrez utiliser le dessin conditionnel (instructions *if* autour du dessin de certaines formes).
* En terminant, assurez-vous que les deux instances placées sur l'application fonctionnent de façon indépendante.
  1. ❓Réflexion : A la place de procéder tel que suggéré au numéro précédent, le programmeur a ouvert la classe Application, sélectionné l'une des instances de *FormesColorees*, puis a ajouté un écouteur de souris. Quelles sont les conséquences négatives de cette approche?

1. Chargez le projet *LigneElastique*. A l'exécution, un pixel rouge apparaît dans le coin supérieur-gauche (une ligne allant de 0,0 à 0,0 !). Modifiez le comportement ainsi: au départ, rien ne doit s'afficher. L'utilisateur fait ensuite un clic n'importe où. Cet endroit sera l'origine fixe de la ligne. Ensuite il déplace la souris, sans tenir le bouton enfoncé. La ligne suit alors le mouvement, toujours tracée entre l'origine fixe de la ligne et la position courante de la souris. La seule façon de cesser ce comportement est de peser sur le bouton *Terminer* (qui quitte l'application).

* Assurez-vous que si l’utilisateur clique une deuxième fois, ce nouvel emplacement représentera le début d’une nouvelle ligne élastique.

1. Chargez le projet *ControleParFleches*.

On désire modifier l'application de manière à ce que les touches de direction (flèches haut, bas, gauche, droite) permettent de déplacer le cercle sélectionné. Utilisez la documentation en-ligne de la classe KeyEvent pour connaître les constantes associées aux touches de flèches. Ces constantes commencent par les caractères "VK\_" (par exemple **KeyEvent.VK\_LEFT**).

* 1. Modifiez l'application pour que les flèches permettent de faire bouger le cercle de quelques pixels à la fois (voir constante DEPLACEMENT), dans le bon dessin, sélectionné par les boutons radio. Étapes:
* Réflexion préalable: quel(s) composant(s) devront posséder un écouteur de clavier? les boutons radio, ou bien le composant de dessin? (c’est à l’enfoncement d touches sur quel composant que l’on veut réagir?). Est-il logique d’encapsuler ce comportement?
* Ensuite, assurez-vous que les boutons radio modifient le fait qu'une ou l'autre des instances du composant de dessin recevra les évènements.
* Ensuite, vous devez vous assurer que c'est le *petitJeu* qui aura le focus au départ. Cette action doit être faite *après* que le composant ait été rendu visible:
  + Dans le *main*, après l'instruction *setVisible*, ajoutez *frame*.*petitJeu.requestFocusInWindow()*;
* Finalement, codez l'écouteur de clavier et effectuez les déplacements nécessaires selon la touche. Contrainte: utiliser une structure *switch*!
  1. Ajoutez le code nécessaire pour que l'application se termine dès que l'utilisateur enfonce la touche q (pour ***q****uitter*).
  2. Le constructeur commence à s'allonger et sa lisibilité se détériore….Pour améliorer la structure de votre code, créez une méthode privée *gererTouchesDuClavier()* qui contiendra toute gestion du clavier, incluant l’instruction *addKeyListener*.
  3. Quand l’utilisateur appuie sur le bouton *Message*, le cercle cesse ensuite de réagir aux touches (car le focus est désormais sur le bouton). Modifiez l’écouteur du bouton de manière à régler ce problème.
  4. ✪Pour les pros : Ajoutez le code nécessaire pour "stopper" le cercle dès qu'il touche à un des quatre côtés de la fenêtre (il doit toutefois encore être possible de le faire bouger dans les autres directions).

1. Chargez le projet *EcouteurSurSaisie*. Vous devez modifier l'application ainsi :
   * Seuls les caractères numériques doivent être permis : les autres provoqueront un "beep" et ne seront affichés nulle part. Utilisez la méthode privée *estUnCaractereNumerique* qui vous est donnée au bas de la classe. Vous aurez aussi besoin de la méthode *consume* vue en classe pour retenir l'apparition du caractère.
   * ✪ Pour les pros : L'application doit répliquer en tout temps dans l'étiquette *lblValeurCourante* le contenu actuel de la zone de saisie.
   * ✪ Pour les pros : Assurez-vous que l’application prévoit que l’utilisateur pourrait effacer certains caractères.
2. ✪ Pour les super pros : Avec l’application réalisée précédemment (cercle jaune), si l’utilisateur enfonce deux flèches *exactement* en même temps, un événement KeyEvent est généré pour chaque touche de flèche et le cercle se déplace selon les deux directions (exemple : droite+bas = diagonale vers le sud-est).

Par contre, si l’utilisateur enfonce d’abord une touche, puis enfonce la deuxième un peu après, l’événement est généré uniquement pour la deuxième touche : en conséquence, le cercle se déplace uniquement dans la direction précisée par la deuxième touche.

Faites une copie du projet précédent. Modifiez-le pour que même si les touches sont enfoncées l’une après l’autre, les deux directions soient prises en considération. Piste de solution: utiliser 4 booléens pour mémoriser la volonté ou non d’appliquer un déplacement dans chaque direction.