ED/TP Génie logiciel, révision

1 Exceptions et révision Java

Les exceptions permettent aux programmes de réagir aux conditions exceptionnelles rencontrées pendant l'exécution.

Soit le code ci-dessous. Essayer d'anticiper son exécution sans le faire tourner réellement. On rappelle qu'un appel `chaine.substring(2)` renvoie le contenu de la chaine `chaine` à partir de la position 2. Exemple : "abcd".substring(2) renvoie "cd".

```
1 class T{
2   //cette méthode divise x en nbSousChaines
3   //et renvoie la dernière sous-chaine
4   //Exemple: diviserChaine("abcdXY",3) devrait renvoier "XY"
5   public static String diviserChaine (String x, int nbSousChaines){
6     try {
7           int pointCoupe = (x.length()/nbSousChaines)*(nbSousChaines-1);
8           return x.substring(pointCoupe);
9     } catch (NullPointerException e) {
10          System.out.println("Problème exécution, je renvoie chaine vide");
11          return "";
12    }
13  }
14  public static void main(String[] args){
15        System.out.println(diviserChaine("abcdefxyztuv",2));
16        System.out.println(diviserChaine(null,1));
17        System.out.println(diviserChaine("abcdef",0));
18        System.out.println(diviserChaine("abcdef",1));
19  }
20 }
```

Exercice 1 On observe que lorsque x n'est pas initialisé, l'appel d'une méthode sur x échoue. C'est pour cette raison qu'on a introduit l'exception `NullPointerException`. Modifier la gestion de cette exception (lignes 10-11) pour faire la fonction renvoyer `null` sans rien afficher.

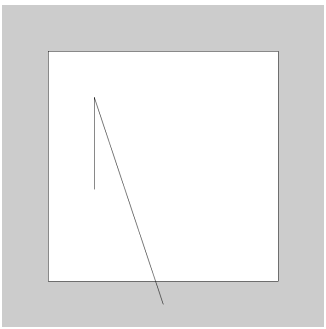
```
System.out.println(diviserChaine("abcdef",0));
```

Exercice 2 On observe que la ligne ci-dessus lève une exception non-gérée. On ne devrait pas demander de diviser la chaîne en 0 sous-chaines. Ajouter un bloc `catch` pour gérer cette exception.

2 Factorisation et Modularité

Soit le code *Processing* :

```
1 boolean ligneInterieurToile(int x1, int y1, int x2, int y2){
2     line(x1,y1,x2,y2);
3     return true;
4 }
5 void setup() {
6     size(700,700);          //taille de fenêtre 700 X 700
7 }
8 void draw() {
9     rect(100,100,500,500);//toile de taille 500 X 500
10    ligneInterieurToile (200,200, 200, 400);
11    ligneInterieurToile (200,200, 350, 650);
12 }
```



Observez que l'une des lignes tracées dépasse le bord de la toile de taille 500×500.

Exercice 1 Modifier la fonction `ligneInterieurToile(...)` pour qu'elle trace une ligne uniquement si elle reste toute entière dans la toile de taille 500 × 500. La fonction devrait renvoyer `true` uniquement s'il est possible de tracer la ligne.

Plusieurs solutions peuvent exister : pensez aux autres, le programme doit être facilement compréhensible

Exercice 2 On considère deux rectangles de même taille `(dimX, dimY)`
- rectangle A situé aux coordonnées `ax, ay` (le coin en haut à gauche)
- rectangle B (obstacle) de coordonnées `bx, by` (le coin en haut à gauche)

Écrire une fonction `intersect(ax, ay, bx, by, dimX, dimY)` qui renvoie `true` si les deux rectangles s'intersectent ou `false` sinon.
Indication : utiliser une fonction (à définir) `pointInterieur(px,py, rx, ry, dimX, dimY)` qui renvoie `true` uniquement si le point/pixel `px,py` est situé à l'intérieur du rectangle de taille `(dimX, dimY)` situé aux coordonnées `rx,ry` (le coin en haut à gauche).

Exercice 3 Modifier complètement l'implémentation de `ligneInterieurToile`. Utiliser juste deux appels `pointInterieur(...)` pour déterminer si la ligne rester toute entière à l'intérieur de la toile.

3 Une animation

On ajoute deux variables globales x et *delta* et on modifie le la méthode `draw` ainsi :

```
int x = 0;
int delta = 1;
boolean ligneInterieurToile(int x1, int y1, int x2, int y2){
    .....
}
void setup() {
    ....
}
void draw() {
    rect(100,100,500,500);      //toile de taille 500 X 500
    ligneInterieurToile (200+x,200, 200+x, 400);
    x+=delta;
}
```

Exercice 1 Ce code permet de déplacer progressivement la ligne à droite. Modifier le code pour que le déplacement change de direction lorsque la ligne "heurte" le mûr. Grâce à la factorisation de code, on doit changer de direction avec un appel comme :

```
if(!ligneInterieurToile(...))
    delta = -delta;
```

On observe que *delta* change de signe, ce que signifie que le déplacement change de direction.

Exercice 2 Modifier le programme pour gérer le déplacement sur les deux axes x et y. Il est conseillé d'utiliser
- une variable *deltaX* et une variable *deltaY* (valeur initiale 1 pour les deux) ;
- Pour changer de direction, il faut choisir entre :
  - $deltaX = -deltaX$ (changement de direction x sans toucher *deltaY*) OU
  - $deltaY = -deltaY$ (changement direction y sans toucher *deltaX*).

On prend le premier choix si
`ligneInterieurToile (200+x+deltaX,200+y, 200+x+deltaX, 400+y)` renvoie `true`.
On prend le deuxième choix si
`ligneInterieurToile (200+x,200+y+deltay, 200+x, 400+y+deltay)` renvoie `true`.