TP : Notion de tableaux et de listes

I) La notion de tableau :

Un tableau constitue la manière la plus efficace pour stocker et accéder aléatoirement à une séquence de données. Chaque donnée est référencée par un index compté à partir de zéro, par exemple, en écrivant:

```
String noms = {"Alice", "Bob", "Samia", "Zheng-You"};
```

nous avons créé une collection de quatre jolis prénoms, et la variable noms[0] a pris la valeur "Alice" et, par exemple, noms[2] a pris la valeur "Samia".

On peut lire une valeur d'un tableau, par exemple:

```
String him = noms[3];
```

donne la valeur "Zheng-You" à la variable de nom him. On peut aussi changer la valeur d'un tableau, par exemple:

```
noms[2] = "Samiha";
```

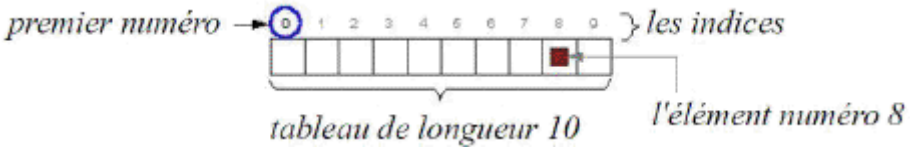
changera la 3ème valeur du tableau, celle d'index 2.

La taille du tableau :

La taille du tableau *noms* est fixée lors de la création et ne peut plus être changé pendant toute la durée de sa vie. La taille d'un tableau est donnée par :

`int longueur = noms.length;` // dans notre cas elle vaut 4

Si on tente d'accéder à une case du tableau qui n'existe pas (par exemple noms[-1] ou noms[4]), alors une erreur est générée lors de l'exécution du programme.



ATTENTION !

Dans un tableau le premier objet est à l'indice 0

Manipuler un tableau de valeurs numériques :

1) Déclaration :

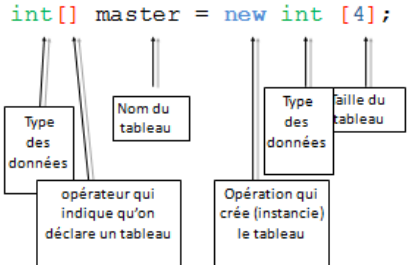
Déclarer un tableau à une dimension :

on utilise l'opérateur [ ]

`int [ ] master;`

déclare un tableau d'entiers nommé master.

Créer le tableau :



Longueur d'un tableau

master.length donne la longueur (nombre d'éléments du tableau) du tableau master

donc l'indice du dernier élément de master est :

master.length - 1

2) Remplissage et/ou affichage d'un tableau : on utilise une boucle

```
for ( int i = 0 ; i < master.length ; i++) {
        // remplissage du tableau avec des entiers aléatoires compris entre 1 et 100
          master [i] = random(1,101) ;
        // affichage des éléments du tableau  en ligne (absence du ln) séparés d'un blanc
          print(master [i] + " ");
}
```

Travail proposé.

Exercice 1 :

Tester ces différentes lignes de code dans la proglet « algoDeMaths » de Javascool ou sur Processing (!!! random(1,101) sera alors remplacé par int(random(1,101)).

Exercice 2 : Trouver l'erreur !!

```
int tab[] = new int[5];
tab[0] = 5;
tab[1] = 9;
tab[2] = "trois";
tab[3] = 59;
```

Le morceau de programme ci-contre est-il correct ? Expliquez pourquoi.

Noter aussi que l'élément tab[4] n'a pas été initialisé c'est à dire que personne n'a donné de valeur à cette case du tableau.
Faire un petit code pour vérifier à quelle valeur Java initialise les valeurs des tableaux par défaut.

Exercice 3 : Appliquer une fonction à un tableau.

Ecrire une fonction

`void double-les-elements(int tab[]) {.....}`

qui prend en argument un tableau tab et multiplie par deux tous ses éléments.

Tableaux à plusieurs dimensions.

Ecrire un algorithme qui remplisse ce tableau à deux dimensions:

```
int [][] table = new int[11][11];
```

avec la table de multiplication, c'est à dire de manière à ce que table[i][j] soit le produit de i par j et afficher le résultat sous forme de table.

Remarques.

1. **Créer et détruire les objets en mémoire.**

Il faut créer un tableau avant de pouvoir l'utiliser, mais en Java pas besoin de se soucier de le détruire quand on ne s'en sert plus: ce sera détecté et géré automatiquement. D'autres langages comme le C/C++ obligent à explicitement détruire les objets qui ne servent plus.

2. **D'autres structures de données**

Il existe des "piles", des "tables", etc. de taille variable, mais c'est une autre histoire…

II) La notion de liste :

Une liste est un tableau dont on ne définit pas la taille
Avec un tableau, on ne peut que consulter ou remplacer un élément d'indice i.
Avec une liste, on peut en plus rajouter un élément de la liste, le supprimer, ……..

1) Déclaration :

- Sur Javascool

Pour commencer, il faut importer la bibilothèque (encore appelée classe) **ArrayList** du package java.util : import java.util.ArrayList ;

```
import java.util.ArrayList;
void main(){
      // création d'une liste nommée maListe de type String
      ArrayList<String> maListe;
      maListe = new ArrayList<String>();
      //.....................................................
}
```

//création d'une liste de caractères	//création d'une liste d'entiers	//création d'une liste de nombres
ArrayList<Character> liste1; liste1=new ArrayList<Character>();	ArrayList<Integer> liste1; liste1=new ArrayList<Integer>();	ArrayList<Float> liste1; liste1=new ArrayList<Float>();

- Sur Processing

On peut utiliser le même codage que sur Javascool mais, depuis Processing 2, il est inutile d'importer la classe **ArrayList**

Depuis la version 2.01, il existe dans Processing les listes : IntList, FloatList et StringList.
Celles-ci sont fournies avec des méthodes spécifiques à Processing…..

// création d'une liste d'entiers	// création d'une liste de nombres	/*création d'une liste de chaînes de caractères*/
IntList liste1 = new IntList() ;	FloatList liste1 = new FloatList() ;	StringList liste1 = new StringList() ;

2) Méthodes pour manipuler les listes

- **Remplissage**

a) Pour ajouter un élément à la liste, on applique la **méthode** add (élément) à l'objet `maListe`.
Par exemple, il suffit d'écrire l'instruction : `maListe.add("Pluton ") ;`

objet.methode() ;

Cela permet d'ajouter `Pluton` (objet de type String) à la liste `maListe`

b) `maListe`.add(i,8) permet d'ajouter 8 à l'élément de la liste `maListe` d'indice i **(spécifique à processing)**

- **La taille d'une liste**

On utilise la méthode size() appliquée à l'objet `maListe`.

- **Affichage de l'élément d'indice i**

L'instruction est : println(maliste.get(i));

- **Trier une liste**

Pour trier une liste, vous devrez importer la classe Collections du package java.util :
import java.util.Collections ;
L'instruction suivante : Collections.sort(maListe) ; trie la liste `maListe`.

- **Remplacer un élément de la liste**

on applique la **méthode** set(i,élément) à l'objet `maListe`.

- **Ajouter un élément dans la liste**

On applique la méthode append(élément) à l'objet `maListe` **(spécifique à processing)**

- **Supprimer un élément de la liste**

on applique la **méthode** remove(i) à l'objet `maListe`.

- **Vider une liste**

on applique la **méthode** clear() à l'objet `maListe`.

Mini-projet 1 (environnement Processing)

L'objectif est de mettre au point un programme qui permet à l'utilisateur de percer des balles qui défilent sur l'écran avec une cible triangulaire que l'on déplace avec les flèches du clavier

Identification des besoins :

1. Une fenêtre assez grande, un fond noir.
2. Des balles (de rayon 30 pixels) qui traversent la fenêtre.
3. Une cible triangulaire que l'on déplace au clavier avec les flèches de déplacement
4. Gérer les collisions entre la cible et les balles.

L'algorithme que nous allons utiliser est :

C'est finalement facile de créer des listes dans Processing....

```
< !----------initialisation------------>
// On crée les deux listes pour contenir les coordonnées des boules
IntList listeX = new IntList();
IntList listeY = new IntList();
// Coordonnées x et y  du centre
//du triangle
int triangleX = 500;
int triangleY = 300;
< !------>
```

Ces variables sont initialisées ici pour pouvoir être utilisées dans n'importe quelle partie du programme

C'est tout !!

```
< !----Traitement--->
void setup(){
//initialisation de la fenêtre :
size(800, 600);
}

void draw(){
background(0); // Fond de la fenêtre (noir)
// On a une chance sur 30 d'ajouter une boule
        if((int)random(0, 30) == 0){
        ajouterBalles();//c'est une fonction à créer
        }
 bougerBalles();//c'est une fonction à créer
 bougerTriangle();//c'est une fonction à créer
 collisionBalles();//c'est une fonction à créer
 affichage();//c'est une fonction à créer
}
```

Le void draw() {...} est une boucle infinie....
A chaque « tour » (la fréquence dépend du frameRate()) il exécute les instructions :

- Remet le fond en noir(permet d'effacer..]
- Ajout de balles (avec un peu d'aléatoire)
- Faire bouger les balles et le triangle
- Gérer les collisions des balles avec le triangle
- Faire afficher le tout

- La fonction ajouterBalles ( ) { …….}

listeX et listeY contiennent les coordonnées du centre des balles.
Cette fonction devra ajouter les coordonnées de la nouvelle balle $(x ; y)$.
Il faut qu'elles apparaissent en dehors de la fenêtre (par exemple à gauche) et de façon aléatoire en hauteur.

- La fonction bougerBalles ( ) { …….}

Nos balles doivent traverser la fenêtre en gardant toujours la même hauteur.
Il s'agit donc d'incrémenter tous les éléments de la listeX de 2 par exemple.

- La fonction bougerTriangle ( ) { …….}

Le code suivant est à compléter (les incrémentations ou décrémentations seront de 3 pour le déplacement de la cible).

```
if(keyPressed){ // Si une touche est appuyée
  if(key == CODED){ // Si c'est une touche "spéciale" (= ni une lettre ni un chiffre)
    if(keyCode == UP){ // Si c'est la flèche du haut

    }
    else if(keyCode == DOWN){ // Etc

    }
    else if(keyCode == LEFT){ // Etc

    }
    else if(keyCode == RIGHT){ // Etc

    }
  }
}
```

- La fonction collisionBalles ( ) { …….}

Cette fonction devra tester si une balle rentre en collision avec la cible et ce pour toutes les balles ….
A l'intérieur d'une boucle qui parcourt la listeX (par exemple), il faudra **tester** si la **distance** entre **le centre de la balle** et les coordonnées de l'un des **sommets du triangle** est **inférieure** au **rayon** de la balle

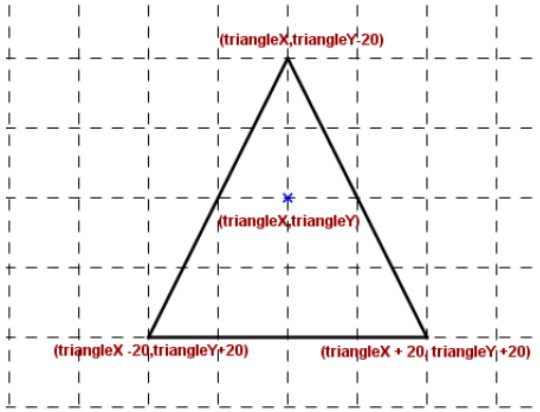
La distance est donnée par la fonction : **dist(listeX.get(i),listeY.get(i),triangleX,triangleY);**

Si le test est positif, il faut supprimer la balle (en supprimant les coordonnées de son centre dans les listes) ….

- La fonction affichage ( ) { …….}

Afficher toutes les balles (qui doivent être blanches) en parcourant la liste (listeX),

Afficher le triangle rouge (aux coordonnées ….)

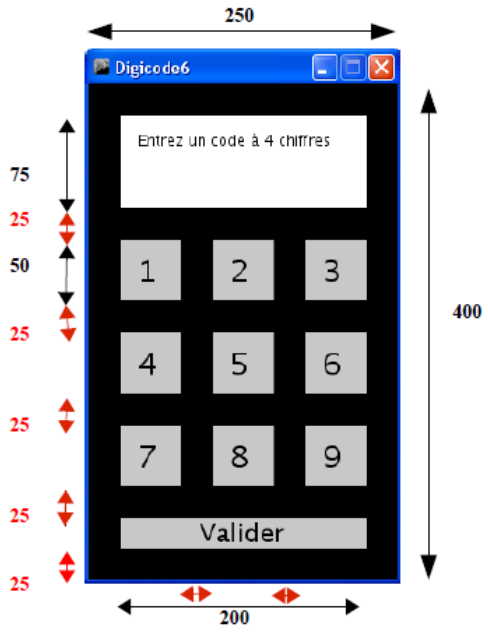


Faire fonctionner ce programme.
Faire afficher le nombre de balles éclatées…. (compteur) sous le forme : **SCORE : ….**
……. Toute autre idée ……. (*ajouter un son WAV à chaque explosion : tutoriel sur la gestion des fichiers audio à l'adresse* http://fr.flossmanuals.net/processing/ch028_la-lecture-du-son)
Ce mini-projet peut être le point de départ d'un projet de BAC (jeu du type space invaders…….)

Mini-projet 2 (environnement Processing)

coder l'interface graphique d'un digicode

Le cahier des charges du digicode



Ce digicode sera représenté dans une fenêtre graphique de dimension 250 x 400 pixels. Il comporte :

- d'un écran alphanumérique de dimension 200 x 75 pixels (le rectangle dans lequel est écrit «Entrez un code à 4 chiffres»)
- de 9 touches numériques de dimension 50 x 50 pixels
- d'une touche «Valider» de dimension 200 x25 pixels

L'écart entre toutes ces « zones » est de 25 pixels.

Le programme à coder

On vous demande de respecter les 7 étapes suivantes qui vous permettront de structurer votre code :

Etape n°1 : l'écran alphanumérique

- Dessinez l'écran du digicode et le texte « Entrez un code à 4 chiffres »
- Dessinez ensuite la touche «Valider» avec son inscription

On utilisera les méthodes **rect**(x1 , y1 , x2 , y2) et **text**(message , x , y)

Etape n°2 : le clavier numérique complet

- Dessiner toutes les touches en utilisant **deux boucles** « **Pour** » imbriquées l'une dans l'autre, de manière à ce que, dans votre code, les méthodes **rect**() et **text**() n'appaissent qu'une seule fois.

Etape n°3 : le nombre de clics de l'utilisateur

- Votre programme doit maintenant utiliser une variable de type entier qui totalisera le nombre de clics de l'utilisateur sur le digicode. Ce nombre devra s'afficher sur l'écran du digicode et cette variable devra pouvoir être utilisée dans tout le programme (variable de portée globale).

On utilisera la méthode **mousePressed(){...}** pour afficher ce nombre de clics.

Etape n°4 : la détection de la touche "1"

- On s'intéresse dans un premier temps à la touche [1]. On veut afficher à l'écran le chiffre 1 lorsqu'on clique sur la touche 1. Plus précisément, si on n'a pas cliqué sur la touche 1 alors l'écran doit afficher « Entrez un code à 4 chiffres », sinon l'écran doit afficher «1 ».
  On utilisera une structure de type : "**Si... Alors...Sinon...**" **: if(...) {…} else {…}**

Les coordonnées de la souris sont obtenues avec les fonctions **mouseX** et **mouseY.** L'opérateur booléen "**et**" est codé **: &&**

Etape n°5 : la détection de toutes les touches

- On s'intéresse maintenant aux autres touches numériques. On veut afficher à l'écran le chiffre sur lequel on clique. Plus précisément, si on n'a pas cliqué sur une touche numérique alors l'écran doit afficher « Entrez un code à 4 chiffres » sinon l'écran doit afficher la touche sur laquelle on a cliqué. Vous devrez également créer une variable booléenne qui prendra la valeur **true** si on a cliqué sur la touche « valider » et **false** dans le cas contraire.

Etape finale : le digicode complet

- Créer un premier tableau codeSecret[] dans lequel vous allez mettre votre code secret à 4 chiffres et un second qui enregistrera les touches frappées (attention, la première valeur d'un tableau est code[0]). Quand l'utilisateur clique sur « valider », on doit comparer les valeurs contenues dans ces deux tableaux. Si les valeurs des deux tableaux sont les mêmes, le message « Vous pouvez entrer » doit s'afficher et ensuite s'affichera votre page web préférée sinon c'est le message « Essayez encore » qui doit s'afficher.
  On utilisera pour cette dernière étape la boucle "**Tant Que ... Faire ...**".

Une boucle "**Tant Que ... Faire**" est codée : **while(...){...}** et le test d'une égalité **A==B**
L'instruction "link(http://.....) ; " permet d'ouvrir la page web appelée.