Yann.Secq@univ-lille.fr
Fabien.Delecroix@univ-lille.fr

Préparation au TD n°4

Objectifs : Consolidation des boucles à compteur. Initiation aux boucles à événement.

Exercice 1 : Ça fait répétition

Voici un programme utilisant une boucle `while`.

```
class ProgrammeWhile extends Program{
  void algorithm(){
    int indice = 0;
    String chaine = "Une phrase composée de plusieurs mots";
    while (indice<length(chaine)/2){
      print(charAt(chaine,indice));
      indice=indice+1;
    }
  }
}
```

1. Quelle est la valeur de `indice` au début du premier tour de la boucle `while` ?
2. Quelle est la valeur de `indice` à la fin du dernier tour de la boucle `while` ?
3. Combien de fois la fonction `print` sera-t-elle appelée lors de l'exécution de ce programme ?
4. Qu'affiche précisément ce programme ?
5. Modifier ce programme pour qu'il s'arrête dès qu'il a rencontré un espace.
6. Dessinez l'algorigramme de ce programme.

Exercice 2 : Ça tourne en boucle

Lors du pré-TD précédent, on était passé du programme de gauche à celui de droite.

```
class ProgrammeRepetitif extends Program {
  void algorithm(){
    int i=1;
    println("affichage numéro "+i);
    i=i+1;
    println("affichage numéro "+i);
    i=i+1;
    println("affichage numéro "+i);
    i=i+1;
    println("affichage numéro "+i);
    i=i+1;
    println("affichage numéro "+i);
  }
}
```

```
class AvecBoucleFor extends Program {
  void algorithm(){
    for (int i=1;i<=5;i=i+1){
      println("affichage numéro "+i);
    }
  }
}
```

Écrivez un programme produisant le même résultat mais en utilisant cette fois une boucle `while`.

Exercice 3 : Contrôle de saisie : ah ces utilisateurs/utilisatrices !

Une tâche récurrente lorsque l'on écrit des programmes nécessitant des saisies consiste à vérifier que l'utilisateur ou l'utilisatrice a bien fait ce que l'on a demandé ... ce qui n'est pas toujours le cas ! Dans le programme ci-dessous, on souhaite saisir un prénom et l'on considère qu'une chaîne non vide peut convenir (discutable, mais soyons simples ;)). Complétez le code ci-dessous afin d'avoir un programme valide.

```
class ControleDeSaisie extends Program {
  void algorithm(){
    boolean saisieInvalide = ...; // à compléter
    String prenom;
    while (saisieInvalide){
```

```
6             print("Veuillez entrer votre prénom : ");
7             // à compléter
8           }
9         }
10      }
```

Serait-il possible de se passer de la variable booléenne ?

Exercice 4 : Potato pirates !

Afin de manipuler les notions abordées depuis le début du cours, nous allons utiliser un autre jeu de cartes permettant d'utiliser ces concepts dans un contexte moins technique. Le but du jeu *Potato pirates* est terriblement simple : couler les bateaux de tous les autres joueurs et joueuses afin de devenir le roi ou la reine des pirates. Pour cela, les participant·e·s définissent de courts algorithmes définissant des attaques sur les bateaux adverses.

Au début du jeu, tous les joueurs et joueuses disposent de deux bateaux avec un équipage de 20 personnes (il faut noter cela à l'aide d'un stylo effaçable sur chacun de vos bateaux) ainsi que 5 cartes.

Pour attaquer les bateaux adversaires, il faut définir des stratégies à l'aide de vos cartes en les disposant en dessous de vos bateaux. Attention : on ne peut utiliser plus de 3 cartes pour définir un algorithme d'attaque et une fois celui-ci définit, il ne sera possible de le déclencher qu'au tour suivant (le temps que vos équipiers s'entraînent à cette nouvelle stratégie !).

Lorsque c'est à votre tour de jouer, vous devez d'abord piocher deux cartes puis :

— définir un ou plusieurs algorithmes d'attaque (autant que de bateaux possédés),

— déclencher une attaque d'un algorithme définit lors d'un précédent tour,

Les cartes *interruption* peuvent être utilisées à n'importe quel moment !

Si suite à une attaque un bateau n'a plus d'équipage, alors il coule et est retiré du jeu. Selon le temps disponible, le jeu s'arrête dès qu'un bateau est coulé (partie courte) ou lorsqu'un des joueuers et joueuses a coulé tous les bateaux des adversaires !

Afin d'avoir des algorithmes un peu plus intéressants (ou violents selon le point de vue), nous nous autoriserons des algorithmes pouvant contenir jusqu'à 5 cartes au maximum ...