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

Objectifs: Utiliser des conditions et des alternatives simples.

Exercice 1 : Écrire des conditions [EXPR-BOOL]

Dans cet exercice vous serez amenés à écrire des conditions qui utilisent des variables déjà définies.

1. Dans un premier temps, les conditions porteront sur des entiers. Copiez le programme ci-dessous et remplacez les valeurs `false` par les conditions correspondant aux commentaires.

```
class ConditionsSurEntiers extends Program {
  void algorithm (){
    int a = readInt();
    int b = readInt();
    int c = readInt();

    // Écrire les conditions à la place des valeurs false

    // Condition a est supérieur à 5.
    boolean a_superieur_a_5 = false ; // MODIFIER false pour définir la bonne condition
    // Condition la somme de a et b est égale à c.
    boolean a_plus_b_egal_c = false ; // MODIFIER false pour définir la bonne condition
    // Condition b est inférieur à a et à c
    boolean b_inferieur_à_a_et_à_c = false ;// À MODIFIER
    // Condition a,b,c vont du plus petit au plus grand
    boolean a_b_c_ordonnes = false ; // À MODIFIER
    // Condition c n'est pas le plus petit des trois nombres
    boolean c_n_est_pas_le_plus_petit = false ; // À MODIFIER


    // Ce qui suit sert à tester vos conditions ; ne pas le modifier

    if (a_superieur_a_5) {
      println(a + "␣est␣supérieur␣à␣5");
    } else {
      println(a + "␣n'est␣pas␣supérieur␣à␣5");
    }

    if (a_plus_b_egal_c) {
      println(a + "+" + b + "=" + c);
    } else {
      println(a + "+" + b + "␣n'est␣pas␣égal␣à␣" + c);
    }

    if (b_inferieur_à_a_et_à_c) {
      println(b + "␣est␣inférieur␣à␣" + a + "␣et␣à␣" + c);
    } else {
      println("Il␣est␣faux␣que␣" + b + "␣est␣inférieur␣à␣" + a + "␣et␣à␣" + c);
    }

    if (a_b_c_ordonnes) {
      println("Il␣est␣vrai␣que␣" + a + "<=" + b + "<=" + c);
    } else {
      println("Il␣n'est␣pas␣vrai␣que␣" + a + "<=" + b + "<=" + c);
    }

    if (c_n_est_pas_le_plus_petit) {
      println(c + "␣n'est␣pas␣le␣plus␣petit␣parmi␣"+a+",␣"+b+",␣"+c);
    } else {
      println(c + "␣est␣le␣plus␣petit␣parmi␣"+a+",␣"+b+",␣"+c);
```

```
    }
  }
}
```

Compilez et exécutez pour vérifier que les phrases affichées sont correctes avec les entrées suivantes :

Entrées à essayer
0, 0, 0
1, 0, 5
2, 4, -6
0, 0, 34
2, 4, 2

2. Dans ce deuxième programme, les conditions vont porter sur des chaînes de caractères.

```
class ConditionsSurChaines extends Program {

  void algorithm (){
    String a = readString();
    String b = readString();
    String c = readString();
     // Écrire les conditions à la place des valeurs false

     // Condition la longueur de a est inférieure à 5
      boolean condLongAinf5 = false ;//À MODIFIER
      // Condition a et b sont la même chaîne
      boolean condAEgalB = false ;//À MODIFIER
      // Condition la première lettre de b précède la première lettre de a d'après l'ord
du dictionnaire
      boolean condBprecedeA = false ;//À MODIFIER
      // (Optionnel) Condition c est un prefixe de a
      boolean condCprefixeDeA = false ;//À MODIFIER
      // Condition la longueur de a est plus grande que celle de c
      boolean condAplusLongueQueC = false ;//À MODIFIER


      // Ce qui suit sert à tester vos conditions; ne pas le modifier

      if (condLongAinf5) {
         println("\"" + a + "\" a moins de 5 caractères");
      } else {
         println("\"" + a + "\" a 5 caractères ou plus");
      }

      if (condAEgalB) {
         println("\"" + a + "\"=\"" + b + "\"");
      } else {
         println("\"" + a + "\" n'est pas égal à \"" + b + "\"");
      }

      if (condBprecedeA) {
         println("\""+b+"\" est avant \"" + a + "\" dans le dictionnaire");
      } else {
         println("\""+b+"\" n'est pas avant \"" + a + "\" dans le dictionnaire");
      }

      if (condCprefixeDeA) {
         println("\"" + c + "\" est préfixe de \"" + a + "\"");
      } else {
         println("\"" + c + "\" n'est pas préfixe de \"" + a + "\"");
      }

      if (condAplusLongueQueC) {
         println("\"" + a + "\" est plus long que \"" + c + "\"");
      } else {
         println("\"" + c + "\" est au moins aussi long que \"" + a + "\"");
      }
```

```
    }


}
```

Compilez et exécutez pour vérifier que les phrases affichées sont correctes avec les entrées suivantes :

Entrées à essayer
"bonjour", "jour", "bon"
"", "aa", "aaa"
"aaa", "aaa", "x"

ATTENTION : le test d'égalité == n'est pas approprié pour comparer deux chaînes, il faut utiliser la fonction `equals` à la place.

Exercice 2 : Même longueur ? [EXPR-BOOL]

On souhaite écrire un programme qui, à partir de deux chaînes de caractères définies comme variables, indique si elles sont de même longueur ou non.

```
class MemeLongueur extends Program{
  void algorithm(){
    ... mot1 = "pomme";
    ... mot2 = "poire";
    ... longueurMot1 = ; //COMPLÉTER L'AFFECTATION
    ... longueurMot2 = ; //COMPLÉTER L'AFFECTATION
    ... memeLongueur = ; //COMPLÉTER L'AFFECTATION
    println("Les deux mots sont de la même longueur : " + memeLongueur);
  }
}
```

1. Copier ce programme et compléter-le par les types adéquats partout où les ... apparaîssent.
2. Compléter ensuite les affectations des variables `longueurMot1`, `longueurMot2` et `memeLongueur` par les bonnes instructions. Faites fonctionner ce programme `iJava`. L'exécution attendue consiste en l'affichage suivant :

```
Les deux mots sont de la même longueur : true
```

3. Changez à présent le deuxième mot pour un autre de taille différente et assurez vous que le programme indique bien `false` dans ce scénario.

Exercice 3 : Blabla [EXPR-BOOL]

On veut repérer les mots constitués d'une suite de lettres qui se répète deux fois comme *tintin*, *rentrent* ou *quinquin*. Écrire un programme `Repetition` qui à partir d'une chaîne de caractères stockée dans une variable indique si elle est une répétition de deux fois la même suite de caractères ou non. Voici deux exemples d'exécutions attendues, sachant que votre programme doit traiter toutes les situations possibles :

```
couscous : true
```

```
zigzag : false
```

ATTENTION : le test d'égalité == n'est pas approprié pour comparer deux chaînes, il faut utiliser la fonction `equals` à la place.

Exercice 4 : Menu en mode texte [ALT-CASC]

Il arrive souvent qu'il soit nécessaire de présenter les différentes fonctionnalités d'un logiciel à l'aide d'un menu. Nous allons dans cet exercice réaliser un menu en mode texte proposant 4 actions à l'utilisateur et lui demandant d'en choisir une. Voici un exemple de l'affichage que votre programme `Menu` doit produire:

```
Bienvenue dans le SuperLogicielDeLanTroisMille

1. Ouvrir un document existant.
2. Créer un nouveau document.
3. Enregistrer le document courant.
4. Quitter ce magnifique logiciel.

Veuillez entrer votre choix: 4
Vous avez choisi: "Quitter ce magnifique logiciel."
```

Exercice 5 : Extraire le suffixe d'une chaîne [ALT-SMPL, EXPR-STR]

Concevez un programme `Suffixe` qui affiche les `nbLettres` derniers caractères d'une `chaine` reçue. Si la chaîne n'est pas suffisamment longue, il faudra retourner la chaîne `"ERREUR"`.

Voici 3 exemples d'exécutions attendues :

```
Entrez une chaîne de caractère : Bonjour
Nombre de lettres de fin souhaitées : 4
Résultat : jour
```

```
Entrez une chaîne de caractère : Au revoir
Nombre de lettres de fin souhaitées : 6
Résultat : revoir
```

```
Entrez une chaîne de caractère : Bonjour
Nombre de lettres de fin souhaitées : 10
Erreur, pas assez de caractères !
```

Exercice 6 : Enlever l'espace en début et fin d'une chaîne [ALT-DEG]

Écrire le programme `NettoyerChaine` qui à partir d'une variable contenant une chaîne retourne la chaîne privée de l'éventuel caractère espace qui se trouverait en début et/ou en fin de la chaîne.

```
class NettoyerChaine extends Program {
   void algorithm () {
     String chaine = "␣Bonjour␣";
     println("Avant␣nettoyage␣:");
     println(">"+chaine+"<");
     //À COMPLÉTER


     println("Après␣nettoyage␣:");
     println(">"+chaine+"<");
   }
}
```

Voici une série d'exemples d'exécutions avec différentes valeurs de la variable `chaine`. Votre programme doit fonctionner pour toutes (à la fois évidemment). Ici, vu que c'est une variable et non une entrée utilisateur qui change, pensez bien à recompiler après chaque modification de votre programme.

```
Avant nettoyage :
> Bonjour <
Après nettoyage :
>Bonjour<
```

```
Avant nettoyage :
> Hello<
Après nettoyage :
>Hello<
```

```
Avant nettoyage :
>bye <
Après nettoyage :
>bye<
```

```
Avant nettoyage :
>  x  <
Après nettoyage :
> x <
```

```
Avant nettoyage :
> <
Après nettoyage :
><
```

```
Avant nettoyage :
><
Après nettoyage :
><
```

Exercice 7 : Le plus petit nombre [ALT-SMPL]

1. Vous devez écrire le programme `PlusPetitNombre` qui affiche le plus petit nombre parmi les deux entrés par l'utilisateur. Voici deux exemples d'exécutions attendues :

```
Entrez deux nombres :
12
7
Le plus petit est 7
```

```
Entrez deux nombres :
-1
-1
Le plus petit est -1
```

Exercice 8 : FizzBuzz [ALT-DEG, ALT-COMB, OPT]

Dans cet exercice vous devrez écrire le programme `FizzBuzz` qui lit un nombre entier entré par l'utilisateur et affiche

— `"fizz"` si le nombre est divisible par 3,
— `"buzz"` si le nombre est divisible par 5,
— `"fizzbuzz"` si le nombre est divisible par 3 et par 5
— le nombre reçu sinon.

Voici quelques exemples d'exécutions attendues :

```
Entrez un nombre : 3
fizz
```

```
Entrez un nombre : 5
buzz
```

```
Entrez un nombre : 15
fizzbuzz
```

```
Entrez un nombre : 8
8
```

1. Quelle condition permet de tester qu'un nombre `n` est divisible par 3 ? par 5 ?
2. Écrire le programme FizzBuzz qui répond aux attentes exprimées ci-dessus.
3. Combien d'alternatives avez-vous utilisé ?
4. Est-ce qu'il y a des conditions qui sont les mêmes entre les différentes alternatives. Réécrire votre fonction pour avoir le moins de conditions qui se répètent.

Exercice 9 : Borne de ciné [ALT-COMB]

Le gérant d'un cinéma a besoin d'un programme de gestion de sa billetterie devant prendre en compte ces règles commerciales :

- le prix de base d'une place de cinéma est de 12€,
- les enfants de moins de 10 ans bénéficient d'un demi-tarif,
- les personnes âgées de moins de 16 ans ainsi que celles ayant plus de 60 ans bénéficient elles d'une réduction de 3€,
- pour les séances bénéficiant de la 3D, il y a un supplément de 2€,
- finalement, si le client est un abonné, alors il bénéficie d'une réduction de 20% sur le prix final (ie. 3D incluse le cas échéant).

Voici deux exemples de traces d'exécutions que doit permettre le programme `Cinema` que vous demande le gérant. Assurez-vous qu'il fonctionne aussi dans d'autres cas de figure (on considérera que l'utilisateur saisit toujours des valeurs correctes).

```
Age du spectateur : 9
Option 3D ? (1 si oui, autre chiffre si non) : 1
Abonné ? (1 si oui, autre chiffre si non) : 0
Coût du billet : 8 euros
```

```
Age du spectateur : 64
Option 3D ? (1 si oui, autre chiffre si non) 0
Abonné ? (1 si oui, autre chiffre si non) 1
Coût du billet : 7.2 euros
```

Prolongements

Exercice 10 : Intervalle vide [ALT-SMPL]

Écrire le programme `Intervalle` qui permet de tester si un intervalle d'entiers, donné par ses deux bornes, est vide ou non. On rappelle que pour qu'un intervalle est vide lorsqu'il ne contient aucune valeur. Par exemple, l'intervalle $[0, 0]$ n'est pas vide vu qu'il contient 0. À contrario, l'intervalle $[3, -5]$ lui est vide car la borne inférieure est plus grande que la supérieure. Voici une base pour ce programme.

```
class Intervalle extends Program {
  void algorithm () {
    int borneInf, borneSup;

    print("Entrez la borne inférieure : ");
    borneInf = readInt();
    println();
    print("Entrez la borne supérieure : ");
    borneSup = readInt();
    println();

    //À COMPLÉTER PAR LA BONNE ALTERNATIVE

    println("cet intervalle est vide.");

    println("cet intervalle n'est pas vide.");

  }
}
```

Copiez et complétez le squelette du programme. Voici 3 exemples d'exécutions attendues :

```
Entrez la borne inférieure : 4
Entrez la borne supérieure : 25
Cet intervalle n'est pas vide.
```

```
Entrez la borne inférieure : 3
Entrez la borne supérieure : -5
Cet intervalle est vide.
```

```
Entrez la borne inférieure : 12
Entrez la borne supérieure : 12
Cet intervalle n'est pas vide.
```

Exercice 11 : Intersection de deux intervalles

Vous devez écrire le programme `Intersection` qui calcule l'intervalle égal à l'intersection de deux intervalles dont les bornes sont donnés par l'utilisateur.

Exemples d'exécutions attendues :

```
Entrez les bornes du premier intervalle : 1
10
Entrez les bornes du second intervalle : 5
15
L'intersection de ces deux intervalles est : [5;10]
```

```
Entrez les bornes du premier intervalle : 4
8
Entrez les bornes du second intervalle : 2
4
L'intersection de ces deux intervalles est : [4;4]
```

```
Entrez les bornes du premier intervalle : 2
4
Entrez les bornes du second intervalle : 5
6
L'intersection de ces deux intervalles est : VIDE
```

On supposera ici que l'utilisateur n'entre jamais d'intervalle vide (cf. exo correspondant).

Exercice 12 : Magic 8-Ball [ALT-COMB]

Proposez un programme `Magic8Ball` qui simule le fonctionnement d'une Magic 8-Ball.