



*Rappels*

*Révision*

*Programmation*

*Page d'accueil*

*API Java*



*Page 1 de 19*

*Plein écran*

*Quitter*

# TP de JAVA – Série 6

## Décisions : Instructions conditionnelles

Université de Mons-Hainaut  
2004 - 2005



## 1. Rappels

- L'instruction `if` permet à un programme d'exécuter différentes actions en fonction du résultat d'une condition.

Rappels

Révision

Programmation

Page d'accueil

API Java



Page 2 de 19

Plein écran

Quitter



Rappels

Révision

Programmation

Page d'accueil

API Java



Page 2 de 19

Plein écran

Quitter

## 1. Rappels

- L'instruction `if` permet à un programme d'exécuter différentes actions en fonction du résultat d'une condition.
- Un bloc d'instructions permet de regrouper plusieurs instructions différentes.



Rappels

Révision

Programmation

Page d'accueil

API Java



Page 2 de 19

Plein écran

Quitter

## 1. Rappels

- L'instruction `if` permet à un programme d'exécuter différentes actions en fonction du résultat d'une condition.
- Un bloc d'instructions permet de regrouper plusieurs instructions différentes.
- Les opérateurs relationnels comparent des valeurs. L'opérateur `==` teste une égalité entre deux valeurs.



Rappels

Révision

Programmation

Page d'accueil

API Java



Page 2 de 19

Plein écran

Quitter

## 1. Rappels

- L'instruction `if` permet à un programme d'exécuter différentes actions en fonction du résultat d'une condition.
- Un bloc d'instructions permet de regrouper plusieurs instructions différentes.
- Les opérateurs relationnels comparent des valeurs. L'opérateur `==` teste une égalité entre deux valeurs.
- Si l'on doit comparer des nombres flottants, il ne faut pas tester l'égalité mais si ceux-ci sont *suffisamment proches*.



Rappels

Révision

Programmation

Page d'accueil

API Java



Page 2 de 19

Plein écran

Quitter

## 1. Rappels

- L'instruction `if` permet à un programme d'exécuter différentes actions en fonction du résultat d'une condition.
- Un bloc d'instructions permet de regrouper plusieurs instructions différentes.
- Les opérateurs relationnels comparent des valeurs. L'opérateur `==` teste une égalité entre deux valeurs.
- Si l'on doit comparer des nombres flottants, il ne faut pas tester l'égalité mais si ceux-ci sont *suffisamment proches*.
- Il ne faut pas utiliser l'opérateur `==` pour comparer des `String`. Utilisez plutôt la méthode `equals`.



Rappels

Révision

Programmation

Page d'accueil

API Java



Page 2 de 19

Plein écran

Quitter

## 1. Rappels

- L'instruction `if` permet à un programme d'exécuter différentes actions en fonction du résultat d'une condition.
- Un bloc d'instructions permet de regrouper plusieurs instructions différentes.
- Les opérateurs relationnels comparent des valeurs. L'opérateur `==` teste une égalité entre deux valeurs.
- Si l'on doit comparer des nombres flottants, il ne faut pas tester l'égalité mais si ceux-ci sont *suffisamment proches*.
- Il ne faut pas utiliser l'opérateur `==` pour comparer des `String`. Utilisez plutôt la méthode `equals`.
- La méthode `compareTo` compare des `String` en utilisant l'ordre *lexicographique*.



Rappels

Révision

Programmation

Page d'accueil

API Java



Page 2 de 19

Plein écran

Quitter

## 1. Rappels

- L'instruction `if` permet à un programme d'exécuter différentes actions en fonction du résultat d'une condition.
- Un bloc d'instructions permet de regrouper plusieurs instructions différentes.
- Les opérateurs relationnels comparent des valeurs. L'opérateur `==` teste une égalité entre deux valeurs.
- Si l'on doit comparer des nombres flottants, il ne faut pas tester l'égalité mais si ceux-ci sont *suffisamment proches*.
- Il ne faut pas utiliser l'opérateur `==` pour comparer des `String`. Utilisez plutôt la méthode `equals`.
- La méthode `compareTo` compare des `String` en utilisant l'ordre *lexicographique*.
- Si une variable objet contient la valeur `null`, elle ne référence aucun objet.



- Des conditions multiples peuvent être combinées grâce aux opérateurs logiques (&&, || et !) pour évaluer des décisions complexes. L'arrangement correct dépend de la logique du problème à résoudre.

*Rappels*

*Révision*

*Programmation*

*Page d'accueil*

*API Java*



*Page 3 de 19*

*Plein écran*

*Quitter*



*Rappels*

*Révision*

*Programmation*

*Page d'accueil*

*API Java*



*Page 3 de 19*

*Plein écran*

*Quitter*

- Des conditions multiples peuvent être combinées grâce aux opérateurs logiques (&&, || et !) pour évaluer des décisions complexes. L'arrangement correct dépend de la logique du problème à résoudre.
- Le type boolean a deux valeurs: true et false.



Rappels

Révision

Programmation

Page d'accueil

API Java



Page 3 de 19

Plein écran

Quitter

- Des conditions multiples peuvent être combinées grâce aux opérateurs logiques (&&, || et !) pour évaluer des décisions complexes. L'arrangement correct dépend de la logique du problème à résoudre.
- Le type boolean a deux valeurs: true et false.
- Une méthode prédicat est une méthode qui retourne une valeur booléenne.



Rappels

Révision

Programmation

Page d'accueil

API Java



Page 3 de 19

Plein écran

Quitter

- Des conditions multiples peuvent être combinées grâce aux opérateurs logiques (&&, || et !) pour évaluer des décisions complexes. L'arrangement correct dépend de la logique du problème à résoudre.
- Le type boolean a deux valeurs: true et false.
- Une méthode prédicat est une méthode qui retourne une valeur booléenne.
- La loi de *De Morgan* permet de simplifier des expressions dans laquelle un opérateur ! est appliqué à des termes joints par un && ou un ||.



Rappels

Révision

Programmation

Page d'accueil

API Java



Page 3 de 19

Plein écran

Quitter

- Des conditions multiples peuvent être combinées grâce aux opérateurs logiques (&&, || et !) pour évaluer des décisions complexes. L'arrangement correct dépend de la logique du problème à résoudre.
- Le type boolean a deux valeurs: true et false.
- Une méthode prédicat est une méthode qui retourne une valeur booléenne.
- La loi de *De Morgan* permet de simplifier des expressions dans laquelle un opérateur ! est appliqué à des termes joints par un && ou un ||.

Les deux formes de la loi de De Morgan :

$!(A \ \&\& \ B)$  est équivalent à  $!A \ || \ !B$

$!(A \ || \ B)$  est équivalent à  $!A \ \&\& \ !B$



Rappels

Révision

Programmation

Page d'accueil

API Java



Page 3 de 19

Plein écran

Quitter

- Des conditions multiples peuvent être combinées grâce aux opérateurs logiques (&&, || et !) pour évaluer des décisions complexes. L'arrangement correct dépend de la logique du problème à résoudre.
- Le type boolean a deux valeurs: true et false.
- Une méthode prédicat est une méthode qui retourne une valeur booléenne.
- La loi de *De Morgan* permet de simplifier des expressions dans laquelle un opérateur ! est appliqué à des termes joints par un && ou un ||.  
Les deux formes de la loi de De Morgan :

$!(A \&\& B)$  est équivalent à  $!A \|\ !B$

$!(A \|\ B)$  est équivalent à  $!A \&\& !B$

- Vous pouvez récupérer le résultat d'une condition dans une variable booléenne.



- La classe `Rectangle2D.Double` est similaire à la classe `Rectangle` mais elle utilise des coordonnées et des longueurs en double précision au lieu de nombres entiers. [Voir API](#).

*Rappels*

*Révision*

*Programmation*

*Page d'accueil*

*API Java*



*Page 4 de 19*

*Plein écran*

*Quitter*



Rappels

Révision

Programmation

Page d'accueil

API Java



Page 4 de 19

Plein écran

Quitter

- La classe `Rectangle2D.Double` est similaire à la classe `Rectangle` mais elle utilise des coordonnées et des longueurs en double précision au lieu de nombres entiers. [Voir API](#).
- La classe `Point2D.Double` représente des points dans le plan dont les coordonnées sont en double précision. [Voir API](#).



Rappels

Révision

Programmation

Page d'accueil

API Java



Page 5 de 19

Plein écran

Quitter

## 2. Exercices de révision

R6.1 : Trouvez les erreurs dans le instructions if suivantes :

```
if quarters > 0 then
    System.out.println(quarter + " quarters");
```



Rappels

Révision

Programmation

Page d'accueil

API Java



Page 5 de 19

Plein écran

Quitter

## 2. Exercices de révision

R6.1 : Trouvez les erreurs dans le instructions if suivantes :

```
if quarters > 0 then
    System.out.println(quarter + " quarters");
```

```
if (1 + x > Math.pow(x,Math.sqrt(2)))
    y = y + x;
```



## 2. Exercices de révision

R6.1 : Trouvez les erreurs dans le instructions if suivantes :

```
if quarters > 0 then
    System.out.println(quarter + " quarters");
```

```
if (1 + x > Math.pow(x,Math.sqrt(2)))
    y = y + x;
```

```
if (x = 1) y++;
else if (x = 2) y = y + 2;
```

Rappels

Révision

Programmation

Page d'accueil

API Java



Page 5 de 19

Plein écran

Quitter



Rappels

Révision

Programmation

Page d'accueil

API Java



Page 5 de 19

Plein écran

Quitter

## 2. Exercices de révision

R6.1 : Trouvez les erreurs dans les instructions if suivantes :

```
if quarters > 0 then
    System.out.println(quarter + " quarters");
```

```
if (1 + x > Math.pow(x,Math.sqrt(2)))
    y = y + x;
```

```
if (x = 1) y++;
else if (x = 2) y = y + 2;
```

```
if (x && y = 0) { x = 1; y = 1; }
```



Rappels

Révision

Programmation

Page d'accueil

API Java



Page 5 de 19

Plein écran

Quitter

## 2. Exercices de révision

R6.1 : Trouvez les erreurs dans les instructions if suivantes :

```
if quarters > 0 then
    System.out.println(quarter + " quarters");
```

```
if (1 + x > Math.pow(x,Math.sqrt(2)))
    y = y + x;
```

```
if (x = 1) y++;
else if (x = 2) y = y + 2;
```

```
if (x && y = 0) { x = 1; y = 1; }
```

```
if (1 <= x <= 10)
    System.out.println(x);
```



*Rappels*

*Révision*

*Programmation*

*Page d'accueil*

*API Java*



*Page 6 de 19*

*Plein écran*

*Quitter*

```
if (s != "nickels" || s != "pennies"  
    || s != "dimes" || s != "quarters")  
    System.out.print("Input error!");
```



Rappels

Révision

Programmation

Page d'accueil

API Java



Page 6 de 19

Plein écran

Quitter

```
if (s != "nickels" || s != "pennies"  
    || s != "dimes" || s != "quarters")  
    System.out.print("Input error!");
```

```
if (input.equalsIgnoreCase("N") || "NO")  
    return;
```



Rappels

Révision

Programmation

Page d'accueil

API Java



Page 6 de 19

Plein écran

Quitter

```
if (s != "nickels" || s != "pennies"  
    || s != "dimes" || s != "quarters")  
    System.out.print("Input error!");
```

```
if (input.equalsIgnoreCase("N") || "NO")  
    return;
```

```
int x = Integer.parseInt(input);  
if (x != null) y = y + x;
```



Rappels

Révision

Programmation

Page d'accueil

API Java



Page 6 de 19

Plein écran

Quitter

```
if (s != "nickels" || s != "pennies"  
    || s != "dimes" || s != "quarters")  
    System.out.print("Input error!");
```

```
if (input.equalsIgnoreCase("N") || "NO")  
    return;
```

```
int x = Integer.parseInt(input);  
if (x != null) y = y + x;
```

```
language = "English";  
if (country.equals("US"))  
    if (state.equals("PR")) language = "Spanish";  
else if (country.equals("China"))  
    language = "Chinese";
```



*Rappels*

*Révision*

*Programmation*

*Page d'accueil*

*API Java*



*Page 7 de 19*

*Plein écran*

*Quitter*

**R6.2** : Expliquez les termes suivants et donnez un exemple pour chacun :

- Expression
- Condition
- Instruction
- Instruction simple
- Instruction composée
- Bloc



Rappels

Révision

Programmation

Page d'accueil

API Java



Page 7 de 19

Plein écran

Quitter

**R6.2** : Expliquez les termes suivants et donnez un exemple pour chacun :

- Expression
- Condition
- Instruction
- Instruction simple
- Instruction composée
- Bloc

**R6.3** : Expliquez la différence entre une suite d'instructions `if ...else ...` et des instructions `if` imbriquées. Donnez un exemple pour chaque.



Rappels

Révision

Programmation

Page d'accueil

API Java



Page 7 de 19

Plein écran

Quitter

**R6.2** : Expliquez les termes suivants et donnez un exemple pour chacun :

- Expression
- Condition
- Instruction
- Instruction simple
- Instruction composée
- Bloc

**R6.3** : Expliquez la différence entre une suite d'instructions `if ...else ...` et des instructions `if` imbriquées. Donnez un exemple pour chaque.

**R6.4** : Donnez un exemple de suite d'instructions `if ...else ...` pour lesquelles l'ordre des tests n'a pas d'importance. Donnez un exemple où l'ordre importe.



Rappels

Révision

Programmation

Page d'accueil

API Java



Page 8 de 19

Plein écran

Quitter

**R6.5** : Parmi les paires de String suivantes, laquelle vient en premier dans l'ordre lexicographique?

- "Tom", "Dick"
- "Tom", "Tomato"
- "church", "Churchill"
- "car manufacturer", "carburetor"
- "Harry", "hairly"
- "C++", " Car"
- "Tom", "Tom"
- "Car", "Carl"
- "car", "bar"



Rappels

Révision

Programmation

Page d'accueil

API Java



Page 9 de 19

Plein écran

Quitter

**R6.6** : Compléter la table de vérité suivante en trouvant les valeurs des expressions booléennes données pour toutes les combinaisons possibles des entrées booléennes  $p$ ,  $q$  et  $r$ .

$p$	$q$	$r$	$(p \ \&\& \ q) \    \ !r$	$!(p \ \&\& \ (q \    \ !r))$
false	false	false		
false	false	true		
false	true	false		
⋮	⋮	⋮		



Rappels

Révision

Programmation

Page d'accueil

API Java



Page 9 de 19

Plein écran

Quitter

**R6.6** : Compléter la table de vérité suivante en trouvant les valeurs des expressions booléennes données pour toutes les combinaisons possibles des entrées booléennes  $p$ ,  $q$  et  $r$ .

$p$	$q$	$r$	$(p \ \&\& \ q) \    \ !r$	$!(p \ \&\& \ (q \    \ !r))$
false	false	false		
false	false	true		
false	true	false		
⋮	⋮	⋮		

**R6.7** : Vrai ou faux?  $A \ \&\& \ B$  est équivalent à  $B \ \&\& \ A$  pour n'importe quelle condition booléenne  $A$  et  $B$ .



Rappels

Révision

Programmation

Page d'accueil

API Java



Page 9 de 19

Plein écran

Quitter

**R6.6** : Compléter la table de vérité suivante en trouvant les valeurs des expressions booléennes données pour toutes les combinaisons possibles des entrées booléennes  $p$ ,  $q$  et  $r$ .

$p$	$q$	$r$	$(p \ \&\& \ q) \    \ !r$	$!(p \ \&\& \ (q \    \ !r))$
false	false	false		
false	false	true		
false	true	false		
$\vdots$	$\vdots$	$\vdots$		

**R6.7** : Vrai ou faux?  $A \ \&\& \ B$  est équivalent à  $B \ \&\& \ A$  pour n'importe quelle condition booléenne  $A$  et  $B$ .

**R6.8** : Expliquez la différence entre

```
s = 0;
if (x > 0) s++;
if (y > 0) s++;
```

et

```
s = 0;
if (x > 0) s++;
else if (y > 0) s++;
```



**R6.9** : Utilisez la loi de De Morgan pour simplifier les expressions booléennes suivantes :

$$\!(x > 0 \ \&\& \ y > 0)$$

Rappels

Révision

Programmation

Page d'accueil

API Java



Page 10 de 19

Plein écran

Quitter



**R6.9** : Utilisez la loi de De Morgan pour simplifier les expressions booléennes suivantes :

$!(x > 0 \ \&\& \ y > 0)$

$!(x \neq 0 \ || \ y \neq 0)$

Rappels

Révision

Programmation

Page d'accueil

API Java



Page 10 de 19

Plein écran

Quitter



**R6.9** : Utilisez la loi de De Morgan pour simplifier les expressions booléennes suivantes :

```
!(x > 0 && y > 0)
```

```
!(x != 0 || y != 0)
```

```
!(country.equals("US") && !state.equals("HI")  
&& !state.equals("AK"))
```

Rappels

Révision

Programmation

Page d'accueil

API Java



Page 10 de 19

Plein écran

Quitter



**R6.9** : Utilisez la loi de De Morgan pour simplifier les expressions booléennes suivantes :

```
!(x > 0 && y > 0)
```

```
!(x != 0 || y != 0)
```

```
!(country.equals("US") && !state.equals("HI")  
  && !state.equals("AK"))
```

```
!(x % 4 != 0 || !(x % 100 == 0 && x % 400 == 0))
```

Rappels

Révision

Programmation

Page d'accueil

API Java



Page 10 de 19

Plein écran

Quitter



Rappels

Révision

Programmation

Page d'accueil

API Java



Page 10 de 19

Plein écran

Quitter

**R6.9** : Utilisez la loi de De Morgan pour simplifier les expressions booléennes suivantes :

```
!(x > 0 && y > 0)
```

```
!(x != 0 || y != 0)
```

```
!(country.equals("US") && !state.equals("HI")  
&& !state.equals("AK"))
```

```
!(x % 4 != 0 || !(x % 100 == 0 && x % 400 == 0))
```

**R6.10** : Expliquez la différence entre l'opérateur == et la méthode equals quand on compare des String.



**R6.11** : Expliquez la différence entre les tests

```
r == s
```

et

```
r.equals(s)
```

où `r` et `s` sont des objets de type `Rectangle`.

Rappels

Révision

Programmation

Page d'accueil

API Java



Page 11 de 19

Plein écran

Quitter



Rappels

Révision

Programmation

Page d'accueil

API Java



Page 11 de 19

Plein écran

Quitter

**R6.11** : Expliquez la différence entre les tests

```
r == s
```

et

```
r.equals(s)
```

où `r` et `s` sont des objets de type `Rectangle`.

**R6.12** : Qu'est-ce qui est faux dans ce test vérifiant si `r` est `null`?  
Que se passe-t-il quand ce code est exécuté?

```
Rectangle r;
```

```
...
```

```
if (r.equals(null))
```

```
    r = new Rectangle(5,10,20,30);
```



**R6.13** : Expliquez comment l'ordre lexicographique de la classe `String` diffère de l'ordre de mots dans un dictionnaire ou dans un botin de téléphone.

*Astuce* : Considérez des chaînes comme `UMH`, `umh.ac.be`, `Boite 21`, c'est-à-dire.

Rappels

Révision

Programmation

Page d'accueil

API Java



Page 12 de 19

Plein écran

Quitter



Rappels

Révision

Programmation

Page d'accueil

API Java



Page 12 de 19

Plein écran

Quitter

**R6.13** : Expliquez comment l'ordre lexicographique de la classe `String` diffère de l'ordre de mots dans un dictionnaire ou dans un botin de téléphone.

*Astuce* : Considérez des chaînes comme `UMH`, `umh.ac.be`, `Boite 21`, c'est-à-dire.

**R6.13** : Expliquez pourquoi il est plus difficile de comparer des nombres flottants que des entiers. Ecrivez un code *Java* qui teste si un entier  $n$  est égal à 10 et si un flottant  $x$  est égal à 10.



Rappels

Révision

Programmation

Page d'accueil

API Java



Page 12 de 19

Plein écran

Quitter

**R6.13** : Expliquez comment l'ordre lexicographique de la classe `String` diffère de l'ordre de mots dans un dictionnaire ou dans un botin de téléphone.

*Astuce* : Considérez des chaînes comme `UMH`, `umh.ac.be`, `Boite 21`, c'est-à-dire.

**R6.13** : Expliquez pourquoi il est plus difficile de comparer des nombres flottants que des entiers. Ecrivez un code *Java* qui teste si un entier  $n$  est égal à 10 et si un flottant  $x$  est égal à 10.

**R6.14** : Donnez un exemple de deux nombres flottants  $x$  et  $y$  tel que `Math.abs(x - y)` est plus grand que 1000, mais où  $x$  et  $y$  sont cependant comparables si l'on tient compte de leur magnitude (voir cours 7, p. 25).



Rappels

Révision

Programmation

Page d'accueil

API Java



Page 13 de 19

Plein écran

Quitter

**R6.15** : Considérez le test suivant pour vérifier si un point est à l'intérieur d'un rectangle.

```
Point2D.Double p = ...
Rectangle2D.Double r = ...
boolean xInside = false;
if (r.getX() <= p.getX() &&
    p.getX() <= r.getX() + r.getWidth())
    xInside = true;
boolean yInside = false;
if (r.getY() <= p.getY() && p.getY() <= r.getY())
    yInside = true;
if (xInside && yInside)
    System.out.println("p is inside the rectangle.");
```

Réécrivez ce code pour éliminer les valeurs explicites true et false, en mettant les valeurs des expressions booléennes dans xInside et yInside.



Rappels

Révision

Programmation

Page d'accueil

API Java



Page 14 de 19

Plein écran

Quitter

### 3. Exercices de programmation

**P6.1** : Ecrivez un programme qui affiche toutes les solutions réelles d'une équation du second degré  $ax^2 + bx + c = 0$ . Demandez les valeurs de  $a$ ,  $b$  et  $c$  à l'utilisateur. Si le discriminant  $\rho$  est négatif, affichez un message statuant qu'il n'y a pas de solutions réelles.

Implémentez une classe `QuadraticEquation` dont le constructeur reçoit les coefficients  $a$ ,  $b$ ,  $c$  de l'équation quadratique. Fournissez des méthodes `getSolution1` et `getSolution2` qui retournent les solutions et une méthode `boolean hasSolutions()` qui retourne `false` si le discriminant est négatif.



Rappels

Révision

Programmation

Page d'accueil

API Java



Page 15 de 19

Plein écran

Quitter

**P6.2** : Ecrivez un programme qui prend une entrée de l'utilisateur représentant une carte selon la notation suivante :

Notation	Signification
A	As
2 ... 10	Valeurs des cartes
V	Valet
D	Dame
R	Roi
C	Carreau
H	Coeur (Heart)
P	Pique
T	Trèfle

Votre programme doit afficher la description complète de la carte.  
Pare exemple,

Entrez la carte:

**RH**

Reine de coeur

(en rouge: ce que rentre l'utilisateur comme information)



Rappels

Révision

Programmation

Page d'accueil

API Java



Page 16 de 19

Plein écran

Quitter

**P6.3** : Ecrivez un programme qui lit trois nombres flottants et qui les affiche triés par ordre croissant. Par exemple :

Entrez trois nombres :

4

9

2.5

Les nombres triés sont :

2.5

4

9



Rappels

Révision

Programmation

Page d'accueil

API Java



Page 16 de 19

Plein écran

Quitter

**P6.3** : Ecrivez un programme qui lit trois nombres flottants et qui les affiche triés par ordre croissant. Par exemple :

Entrez trois nombres:

4

9

2.5

Les nombres triés sont:

2.5

4

9

**P6.4** : Ecrivez un programme qui affiche la question “*Voulez-vous continuer?*” et lit ce que l'utilisateur entre. Si l'utilisateur entre "O", "Oui", "OK", "Sur" ou "Pourquoi pas?", affichez "OK". Si l'utilisateur entre "N" ou "Non", affichez "Aurevoir". Autrement, affichez "Mauvaise entree". La casse de l'entrée ne doit pas intervenir. Par exemple, "o" ou "oui" sont des entrées valides. Ecrivez une classe `InputChecker` dans ce but.



Rappels

Révision

Programmation

Page d'accueil

API Java



Page 17 de 19

Plein écran

Quitter

**P6.5** : Ecrivez un programme qui traduit une cote alphabétique (sous forme de lettre) en une cote chiffrée. Les cotes alphabétiques sont A B C D F, éventuellement suivies de + ou -. Leur valeurs numériques sont respectivement 4,3,2,1,0. Il n'y a pas de F+ ou F-. Un + augmente la valeur de 0.3, un - la diminue de 0.3. Cependant A+ vaut 4.

Entrez une cote alphabétique:

B-

La valeur numerique est 2.7.

Utilisez une classe Grade avec une méthode `getNumericGrade`.



Rappels

Révision

Programmation

Page d'accueil

API Java



Page 17 de 19

Plein écran

Quitter

**P6.5** : Ecrivez un programme qui traduit une cote alphabétique (sous forme de lettre) en une cote chiffrée. Les cotes alphabétiques sont A B C D F, éventuellement suivies de + ou -. Leur valeurs numériques sont respectivement 4,3,2,1,0. Il n'y a pas de F+ ou F-. Un + augmente la valeur de 0.3, un - la diminue de 0.3. Cependant A+ vaut 4.

Entrez une cote alphabétique:

B-

La valeur numerique est 2.7.

Utilisez une classe `Grade` avec une méthode `getNumericGrade`.

**P6.6** : Ecrivez un programme qui traduit un nombre entre 0 et 4 en la cote alphabétique la plus proche. Par exemple, le nombre 2.8 (qui peut être la moyenne de plusieurs cotes) doit être converti en B-. Résolvez les cas intermédiaires en faveur de l'étudiant (cote vers le haut). Par exemple, 2.85 vaut B.



Rappels

Révision

Programmation

Page d'accueil

API Java



Page 18 de 19

Plein écran

Quitter

**P6.7** : Ecrivez un programme qui lit 4 chaînes de caractères et affiche la plus petite et la plus grande au sens lexicographique :

Entrez quatre chaînes :

Charlie

Able

Delta

Baker

La chaîne minimum est Able

La chaîne maximum est Delta

*Astuce* : Utilisez une classe qui retient les chaînes minimum et maximum courantes.



Rappels

Révision

Programmation

Page d'accueil

API Java



Page 18 de 19

Plein écran

Quitter

**P6.7** : Ecrivez un programme qui lit 4 chaînes de caractères et affiche la plus petite et la plus grande au sens lexicographique :

Entrez quatre chaînes :

Charlie

Able

Delta

Baker

La chaîne minimum est Able

La chaîne maximum est Delta

*Astuce* : Utilisez une classe qui retient les chaînes minimum et maximum courantes.

**P6.8** : Ecrivez un programme qui demande à l'utilisateur d'entrer un mois (1 = Janvier, 2 = Février, ...) et affiche le nombre de jours de ce mois. Pour février, affichez 28 jours.

Entrez un mois :

5

31 jours

Implémentez une classe `Month` avec une méthode `getDays()`.



Rappels

Révision

Programmation

Page d'accueil

API Java



Page 19 de 19

Plein écran

Quitter

**P6.9** : Ecrivez un programme qui lit deux nombres flottants et teste s'ils sont les mêmes quand on les arrondit à deux décimales près.

Exemples :

Entrez deux nombres flottants:

2.0

1.99998

Ils sont identiques a deux decimales pres.

Entrez deux nombres flottants:

2.0

1.98999

Ils sont differents.



Rappels

Révision

Programmation

Page d'accueil

API Java



Page 19 de 19

Plein écran

Quitter

**P6.9** : Ecrivez un programme qui lit deux nombres flottants et teste s'ils sont les mêmes quand on les arrondit à deux décimales près.

Exemples :

Entrez deux nombres flottants:

2.0

1.99998

Ils sont identiques à deux décimales près.

Entrez deux nombres flottants:

2.0

1.98999

Ils sont différents.

**P6.10** : Améliorez la classe `BankAccount` de la série 3 en

- rejetant les montants négatifs dans les méthodes `deposit` et `withdraw`;
- rejetant les retraits qui provoqueraient un solde négatif.