OC Informatique 20 – 21

GYMNASE DE BURIER

Table des matières

1 Greenfoot

1.1 Concepts de base

Exercice 1.1

On voit ci-dessous le menu contextuel associé à un wombat du scénario leaves-and-wombats.

a) Établir une liste de toutes les méthodes du `Wombat`.

b) Dans la liste de la question a), chercher toutes les méthodes sans valeur de retour.

c) Dans la liste de la question a), chercher toutes les méthodes qui renvoient une valeur. Donner également le type de la valeur de retour.

d) Décrire précisément chaque mot figurant dans l'en-tête de la méthode `setDirection`.

e) Donner la liste de toutes les valeurs que peut renvoyer la méthode `canMove`.

f) Donner quelques exemples de valeurs que peut renvoyer la méthode `getLeavesEaten`.

g) Quelle est la méthode de la classe `Wombat` qui demande un paramètre ? Quel est le type de ce paramètre ?

h) Que signifie le mot-clef `void` ?

Exercice 1.2

On donne le scénario `cars`, dans lequel on trouve la classe `Car`. Ouvrir ce scénario.

a) Est-ce qu'un objet de type `Car` existe à l'ouverture du scénario ?

b) Est-ce que la classe `Car` existe à l'ouverture du scénario ?

c) Combien d'objets de type `Car` peut-on créer dans ce scénario ?

d) Créer quelques voitures. Ces voitures sont-elles différentes ?

e) Cliquer le bouton Act. Comment interpréter le résultat qui s'affiche à la console ?

f) Remettre le scénario à zéro en cliquant le bouton Reset. Créer une voiture dans la première colonne. À l'aide d'un appel à la bonne méthode, faire se déplacer la voiture de deux cases vers la droite.

Exercice 1.3

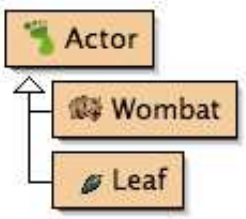
Rédiger l'en-tête d'une méthode publique nommée `send`, disposant d'un paramètre de type `String`, et qui ne renvoie pas de valeur.

Exercice 1.4

Rédiger l'en-tête d'une méthode publique nommée `moyenne`, disposant de deux paramètres, tous deux de type `int`, et qui renvoie une valeur `int`.

Exercice 1.5

Voici le diagramme de classe des acteurs du scénario leaves-and-wombats



On donne également les en-têtes des classes figurant dans ce diagramme à l'exception de la classe `Actor`.

```
public class Wombat extends Actor

public class Leaf extends Actor
```

Dessiner le diagramme de classe des acteurs du scénario asteroids à partir des en-têtes de classes donnés ci-dessous :

```
public class Explosion extends Actor

public class Mover extends Actor

public class Bullet extends Mover

public class Rocket extends Mover

public class Asteroid extends Mover
```

Exercice 1.6

Dessiner un diagramme de classe à partir des concepts suivants :

a) humain, footballeur, avant-centre, sportif, skieur, spécialiste du slalom géant ;

b) guitare, instrument de musique, trompette, instrument à vent, instrument à corde, violon, saxophone, voix.

Exercice 1.7

Observez les en-têtes de méthodes ci-dessous :

```
public void play()
public void addAmount(int amount)
public boolean hasWings()
public void compare(int x, int y, int z)
public boolean isGreater (int number)
```

Pour chaque en-tête, répondez aux questions suivantes :

a) Quel est le nom de la méthode ?

b) La méthode retourne-t-elle une valeur ?
Si c'est le cas, quelle est le type de la valeur de retour ?

c) Combien de paramètres a la méthode ?

Exercice 1.8

Écrivez l'en-tête d'une méthode dont le nom est « `go` ». La méthode n'a pas de paramètre et ne retourne aucune valeur.

Exercice 1.9

Écrivez l'en-tête d'une méthode dont le nom est « `process` ». La méthode a un paramètre de type `int` intitulé « `number` » et elle retourne une valeur de type `int`.

Exercice 1.10

Écrivez l'en-tête d'une méthode dont le nom est « `isOpen` ». La méthode n'a pas de paramètre et retourne une valeur de type `boolean`.

Exercice 1.11

Écrivez un appel de méthode pour la méthode `play` de l'exercice 1.7. Donnez un appel de méthode pour la méthode `addAmount`, ainsi qu'un appel de méthode pour la méthode `compare` du même exercice.

1.2 Gros chat

Tous les exercices qui suivent sont destinés à être faits dans le scénario **Greenfoot : fatcat**. Ouvrez le scénario dans **Greenfoot** avant de poursuivre (on le trouve dans le dossier *book-scenarios/chapter02-04*)

Exercice 1.12

Ouvrez le code la classe `Cat` dans l'éditeur. Assurez-vous que l'éditeur est en mode de *documentation*. De combien de méthodes dispose la classe `Cat` ?

Exercice 1.13

Combien de méthodes de la classe `Cat` retournent une valeur ?

Exercice 1.14

Combien de paramètres contient la méthode `sleep` ?

Exercice 1.15

Essayez d'appeler certaines méthodes de votre chat de façon interactive, en utilisant le menu contextuel du chat. Les méthodes intéressantes proviennent toutes de *Inherited from Cat*.

Exercice 1.16

Est-ce que le chat s'ennuie ? Comment pouvez-vous faire pour qu'il ne s'ennuie pas ?

Exercice 1.17

Ouvrez l'éditeur pour la classe `MyCat`. (C'est là que vous allez écrire le code pour tous les exercices suivants.)

Faites manger le chat quand il agit. (C'est à dire : Dans la méthode `act`, écrire un appel à la méthode `eat`.) Compiler. Testez en appuyant sur le bouton Act dans le panneau de contrôle de l'exécution.

Exercice 1.18

Faites danser le chat. (Ne pas le faire de façon interactive - écrivez du code dans la méthode `act` pour ce faire. Lorsque vous avez terminé, cliquez sur le bouton Act dans le panneau de contrôle de l'exécution.)

Exercice 1.19

Faites dormir le chat.

Exercice 1.20

Faites exécuter à votre chat une séquence d'actions de votre choix, choisies parmi celles qui sont disponibles.

Exercice 1.21

Changer la méthode `act` de votre chat afin que, lorsque vous cliquez Act, si le chat est fatigué, il dort un peu. Si il n'est pas fatigué, il ne fait rien.

Exercice 1.22

Changer la méthode `act` de votre chat pour qu'il danse s'il s'ennuie. (Mais seulement s'il s'ennuie.)

Exercice 1.23

Changer la méthode `act` de votre chat pour qu'il mange s'il a faim.

Exercice 1.24

Modifier la méthode `act` de votre chat de la façon suivante : Si le chat est fatigué, il dort un peu, et puis il crie *hooray*. S'il n'est pas fatigué, il crie seulement *hooray*. (Pour les essais, faire en sorte que le chat soit fatigué en appelant des méthodes de manière interactive. Comment procéder ?)

Exercice 1.25

Modifier le code de la méthode `act` de façon à obtenir le comportement suivant : Si votre chat est seul, faites-le dormir. S'il n'est pas seul, faites le crier : *hooray*. Testez en plaçant un deuxième chat dans le monde avant de lancer Act.

Exercice 1.26

Écrire la méthode `act` de votre chat en respectant les deux points suivants simultanément :

a) si le chat s'ennuie, il danse et crie « Hooray » ;

b) s'il ne s'ennuie pas, il se déplace à gauche de cinq unités et, dans ce cas seulement, s'il est fatigué, il dort durant trois unités de temps.

1.3 Stickman

Les concepts que nous voulons renforcer ici sont les instructions `if` (à nouveau), la lecture de la documentation de l'API, les appels de méthodes depuis une autre classe, et la création de nos propres méthodes.

Pour le faire, nous allons utiliser un autre scénario : stickman. Trouvez-le dans les scénarios du livre et ouvrez-le.

Exercice 1.27

La classe `Greenfoot` dispose d'une méthode qui donne le niveau sonore du microphone intégré de l'ordinateur. Trouvez cette méthode dans la documentation de l'API. Combien de paramètres doivent être donnés lors d'un appel à cette méthode ?

Exercice 1.28

Quel est le type de valeur de retour de cette méthode ? Quelle information nous fournit-elle ?

Exercice 1.29

Quelles sont toutes les valeurs de retour possibles pour cette méthode ?

Exercice 1.30

Cette méthode est-elle *statique* ou non ? Quelle information peut-on en tirer ?

Exercice 1.31

Comment peut-on appeler cette méthode ? Ecrire un appel à cette méthode en utilisant la syntaxe correcte.

Exercice 1.32

Dans votre scénario stickman, faire bouger le personnage vers la droite, de sorte que lorsque vous faites tourner votre scénario, il se déplace jusqu'au bord droit de l'écran.

Exercice 1.33

En utilisant une instruction `if` et la méthode qui permet de capter les sons à l'aide du micro trouvée plus haut, faites en sorte que le personnage ne bouge que lorsque vous faites du bruit. Essayez différentes valeurs du paramètre (qui représente le niveau sonore). Le résultat dépendra de votre micro et de l'environnement dans lequel vous travaillez. Un point de départ raisonnable est de faire bouger le personnage lorsque le niveau sonore perçu par le micro est supérieur à 3. Testez.

Exercice 1.34

Faites en sorte que le personnage bouge d'une part vers la gauche lorsque vous faites du bruit, et d'autre part, continûment vers la droite lorsqu'il n'y a pas de bruit. Testez. Essayez de maintenir le

personnage au centre en faisant du bruit.

Exercice 1.35

Déplacez le code qui provoque le déplacement vers la gauche lorsque vous faites du bruit dans une nouvelle méthode. Nommez cette méthode `moveLeftIfNoise`. Testez. Vérifiez que tout fonctionne comme avant.

Exercice 1.36

Ajoutez une nouvelle méthode à votre scénario qui s'appelle `gameOver`. Cette méthode doit être appelée si le personnage atteint le bord de l'écran. Lorsqu'on l'appelle, la méthode fait jouer un son de fin de jeu et stoppe l'exécution du scénario.

Exercice 1.37

Déplacer la vérification de la condition de fin de jeu dans une méthode séparée appelée `checkGameOver`. La méthode `act` doit appeler `checkGameOver`, qui, à son tour, appelle `gameOver` si nécessaire.

Exercice 1.38

Faire en sorte que le personnage flotte vers le haut lorsqu'il y a du bruit. La hauteur dont il s'élève doit être proportionnelle au niveau sonore. Il redescend s'il n'y a pas de bruit.

Notons que l'exigence concernant le facteur de proportion entre le niveau sonore et la hauteur complique le problème ; vous devrez chercher et utiliser des techniques dont nous n'avons pas encore parlé. Si vous trouvez cela trop difficile, vous pouvez laisser tomber.

Exercice 1.39

Intégrez un nouvel acteur (un animal, par exemple) qui apparaît à gauche de l'écran et qui se déplace latéralement. S'il atteint de bord droit de l'écran, il est déplacé à gauche par le programme. La trajectoire horizontale de cet acteur doit être située à la hauteur du personnage. L'acteur doit « toucher » le personnage lorsqu'il se déplace et tant que le personnage n'a pas pris de hauteur. Il doit être possible de faire sauter le personnage par dessus le nouvel acteur en faisant du bruit.

Exercice 1.40

Faites stopper le scénario (avec le son de fin de jeu) lorsque le personnage touche l'animal.

1.4 Pour aller plus loin

Exercice 1.41

Dans le contexte de programmation de Greenfoot, écrire une instruction conditionnelle dont le corps s'exécute dans 17% des cas.

Exercice 1.42

Réaliser le constructeur de la classe voiture, initialisant la vitesse à 0. Surchargez ce constructeur si l'on connaît la vitesse initiale.

Exercice 1.43

Répondre aux questions indépendantes suivantes.

a) La classe `Avion` possède le constructeur `Avion(int nbrePassagers)`.

Écrire l'instruction permettant de créer un objet appelé `boeing` de type `Avion` avec 100 passagers.

b) Dans la classe `Actor`, on a créé une sous-classe `MyCat`. On veut ajouter dans la classe `MyCat` la méthode privée `tourner(int n)` qui permet d'ajouter `n` degrés à la direction d'un objet du type `MyCat`. Ecrire l'entête et le corps de cette méthode, puis utiliser cette méthode dans le corps de la méthode publique `act()` qui permet d'obtenir le comportement suivant d'un objet de type `MyCat` : s'il se trouve au bord, il tourne à droite par rapport à sa direction d'un angle de 70° et si ce n'est pas le cas, il avance de 3 pas et tourne ensuite à gauche par rapport à sa direction d'un angle de 30°.

c) On a défini et initialisé deux variables `nb1` et `nb2`. Ecrire la séquence d'instructions telle que, suite à ces instructions, `nb2` ait la valeur double de l'ancienne valeur de `nb1` et `nb1` la valeur triple de l'ancienne valeur de `nb2`.

d) La classe `Family` possède la variable `children` de type `int`. Ecrire l'entête et le corps d'une méthode publique `naissance(int n)` qui modifie la variable `children` lors de la naissance de `n` enfants.

e) Dans une classe qui peut utiliser les méthodes de la classe `Greenfoot`, déclarer et initialiser une variable `nombre` qui prend une valeur entière aléatoire entre -20 et 50.

f) Dans la classe `Actor`, on a créé une sous-classe `Lune`. Dans cette sous-classe, à l'aide du fichier `lune1.png` on a créé un objet de type `GreenfootImage` nommé `lune1` qui est utilisé comme image d'un objet de type `Lune`. On dispose d'une deuxième image de lune dans le fichier `Lune2.png`.

Créer et initialiser un objet de type `GreenfootImage` à l'aide du fichier `Lune2.png`, puis écrire l'entête et le corps de la méthode privée `switchImage()` qui permet à un objet de type `Lune` de faire tourner en boucle les deux images avec changement uniquement toutes les 5 méthodes `act`.

Exercice 1.44

On donne ci-dessous la déclaration de la classe Bidon :

```
public class Bidon {
    private int bidon;
    public Bidon(int bidon) {
        this.bidon = bidon;
    }
}
```

a) Quel est le rôle du mot-clef `this` ?

b) Dans quel contexte peut-on écrire l'instruction

```
bidon += 1;
```

vu la façon dont cette variable d'instance est déclarée ?

Exercice 1.45

Dans Greenfoot, créer un scénario Java et, à l'aide du menu Edit, importer la classe `Counter`. On donne ci-dessous le code de la classe `MyWorld` :

```
import greenfoot.*;

public class MyWorld extends World
{
    Counter score;
    public MyWorld() {
        super(600, 400, 1);
        prepare();
    }
    public void act() {
        // À compléter
    }
    private void prepare() {
        // À compléter
    }
}
```

Compléter le code de la classe de sorte qu'un compteur s'affiche dans le monde et qu'il montre le nombre d'appels à la méthode `act` lors de l'exécution du scénario.

Exercice 1.46

On donne ci-dessous la déclaration de la classe Bacteria :

```
public class Bacteria extends Actor
{
    private int speed;
    public Bacteria() {
        speed = Greenfoot.getRandomNumber(5) + 1;
    }
    public void act() {
        setLocation(getX()-speed, getY());
        turn(1);
        if (getX() <= 0) {
            Bloodstream bloodstream = (Bloodstream)getWorld();
            bloodstream.addScore(-15);
            bloodstream.removeObject(this);
        }
    }
}
```

a) Quelle est l'instruction qui attribue une vitesse à une bactérie ?

b) Ecrire l'instruction qui permet d'ajouter à la classe `Bacteria` un champ de type entier nommé `verticalSpeed`.

c) Ecrire l'instruction qu'il faudrait placer dans le constructeur pour donner aléatoirement une valeur entière comprise entre -1 et 1 à la variable `verticalSpeed`.

d) Ecrire l'instruction qu'il faudrait placer dans la méthode `act` pour tenir compte de la composante verticale de la vitesse de la bactérie, donnée par la valeur de `verticalSpeed`.

Exercice 1.47

Dans la classe `WhiteCell` dont le code est donné plus bas, on dispose d'une variable `score`, de type entier. Écrire dans cette classe une méthode `updateScore` qui permet d'ajouter un certain nombre à la variable `score`. Modifier ensuite le code de cette classe de sorte que le score augmente de 10 chaque fois que la méthode `act` a été appelée 20 fois. Si nécessaire, ajouter un champ à la classe `WhiteBloodCell`.

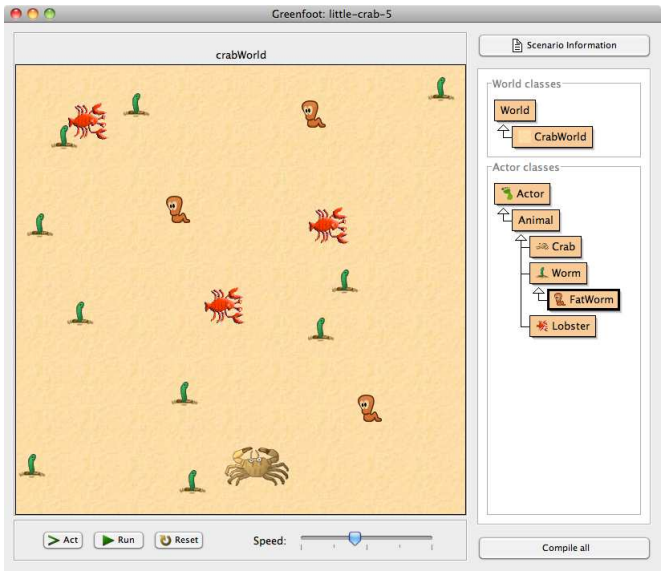
```
import greenfoot.*;
public class WhiteCell extends Actor
{
    private int score;

    public void act()
    {
        checkKeyPress();
    }
    private void checkKeyPress()
    {
        if (Greenfoot.isKeyDown("up"))
        {
            setLocation(getX(), getY()-4);
        }

        if (Greenfoot.isKeyDown("down"))
        {
            setLocation(getX(), getY()+4);
        }
    }
}
```

Exercice 1.48

On a ajouté au scénario little-crab-5 une sous-classe `FatWorm` de la classe `Worm`.



On donne ci-dessous le code source de la méthode `lookForWorm` de la classe `Crab` :

```
/**
 * Check whether we have stumbled upon a worm.
 * If we have, eat it. If not, do nothing. If we have
 * eaten eight worms, we win.
 */
public void lookForWorm()
{
    if ( isTouching(Worm.class) )
    {
        removeTouching(Worm.class);
        Greenfoot.playSound("slurp.wav");
        wormsEaten = wormsEaten + 1;
        if (wormsEaten == 8)
        {
            Greenfoot.playSound("fanfare.wav");
            Greenfoot.stop();
        }
    }
}
```

L'appel de méthode `isTouching(Worm.class)` retourne `true` si le crabe voit un objet de la classe `Worm` ou de sa sous-classe `FatWorm`.

Modifier la méthode `lookForWorm` de façon à ce que l'attribut `wormsEaten` soit augmenté de 2 lorsque le crabe mange un objet de la classe `FatWorm`, toutes choses restant égales par ailleurs.

Réécrire complètement la méthode `lookForWorm` ci-dessous, après lui avoir fait les modifications nécessaires. On veillera à changer le commentaire...

1.5 Scénarios Greenfoot

Exercice 1.49 (WaterFall)
Créer le scénario WaterFall pour obtenir un « jeu » analogue à celui présenté dans la vidéo waterFall.mp4 donnée en annexe. Suivre les indications ci-dessous :

a) Le monde WaterWorld doit être formé de 800×600 cellules de dimensions 1×1 pixels.

b) Après compilation, le seau apparaît en bas de l'écran, à 500 pixels du haut du monde. Il est centré horizontalement.

c) Le seau ne se déplace que latéralement ; pour modifier sa position, on utilise les flèches « gauche » et « droite ».

d) Les gouttes apparaissent au plafond à intervalles irréguliers, à 50 pixels du haut du monde et sont réparties aléatoirement sur la largeur du monde.

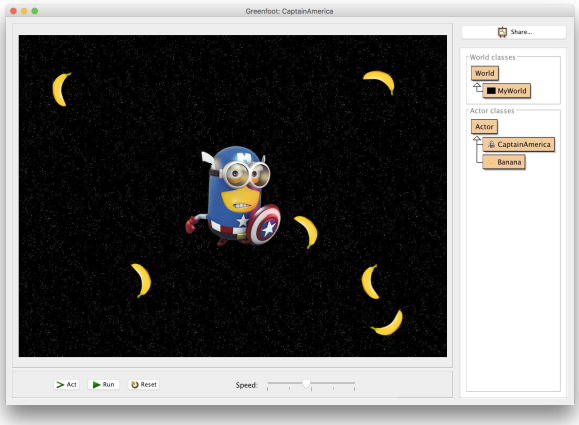
e) La vitesse de chute d'une goutte est constante, mais aléatoire.

f) Un son est joué lorsqu'une goutte « tombe » dans le seau ; il est donné dans les ressources du scénario (watersound.wav).

g) Les gouttes disparaissent dès qu'elles se trouvent à plus de 590 pixels du haut du monde ou lorsqu'elles tombent dans le seau.

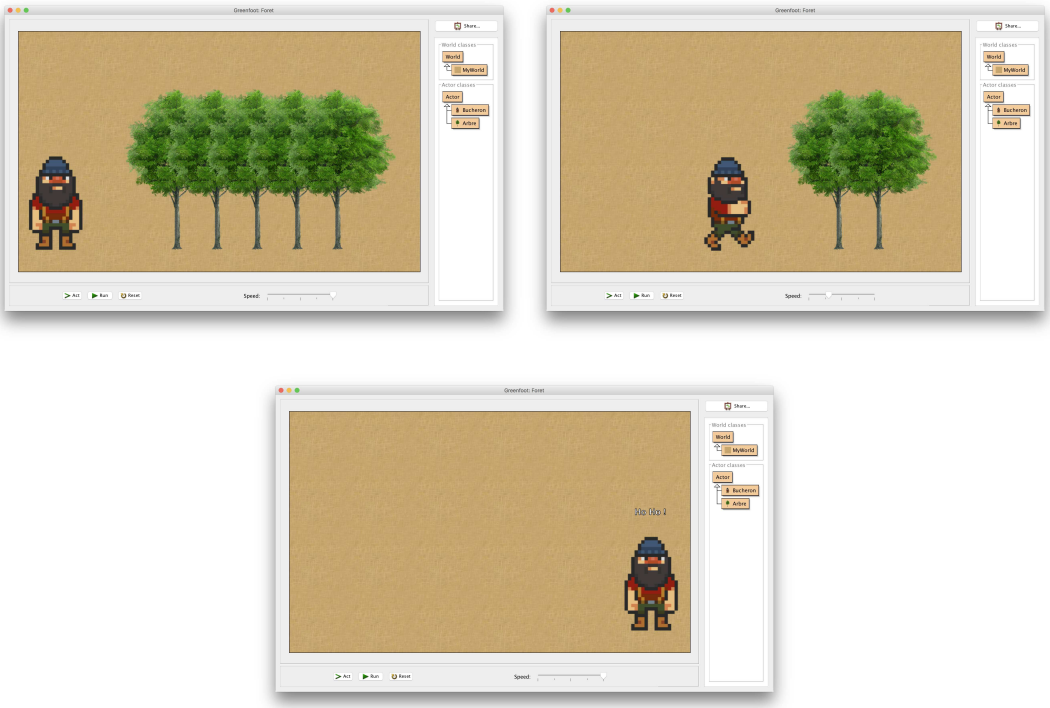
Exercice 1.50 (Captain America)

Compléter le scénario `CaptainAmerica` donné pour obtenir un résultat similaire à celui présenté dans la video `Captain.mov`.



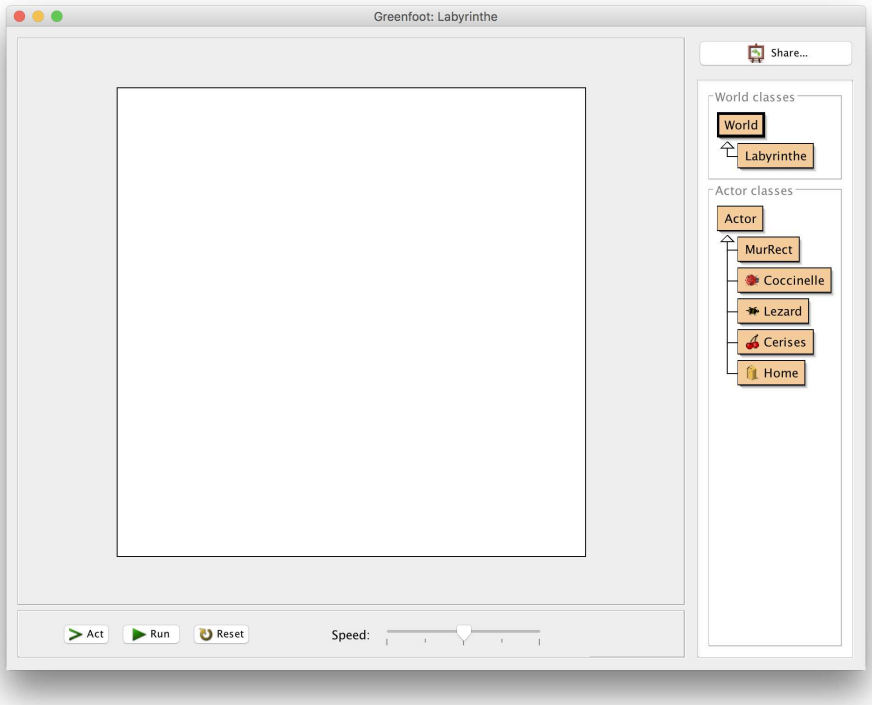
Exercice 1.51 (La Fôret)

Compléter le scénario `Foret` donné pour obtenir un résultat similaire à celui présenté dans la video `Foret.mov`.



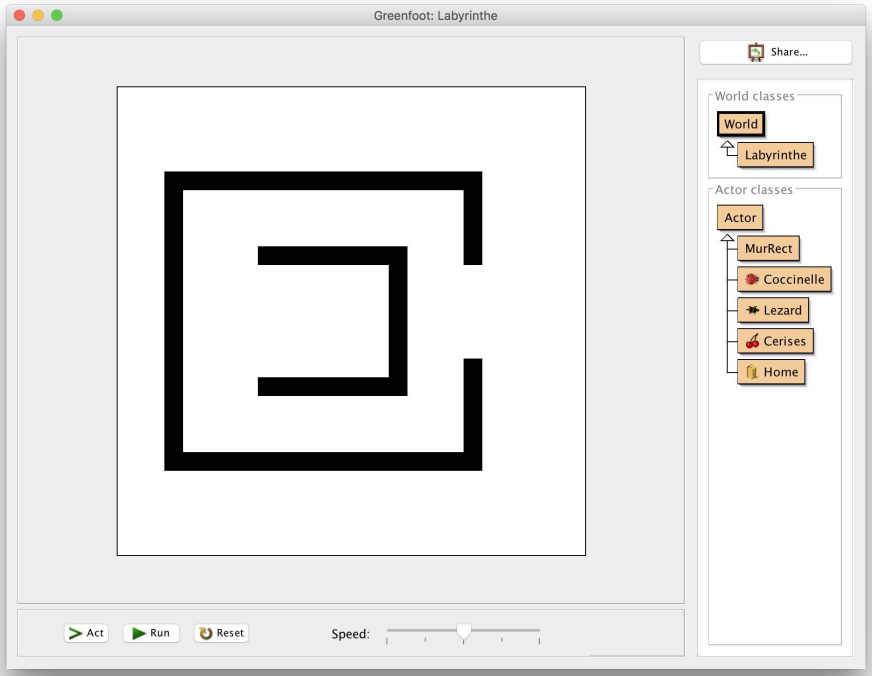
Exercice 1.52 (Labyrinthe)

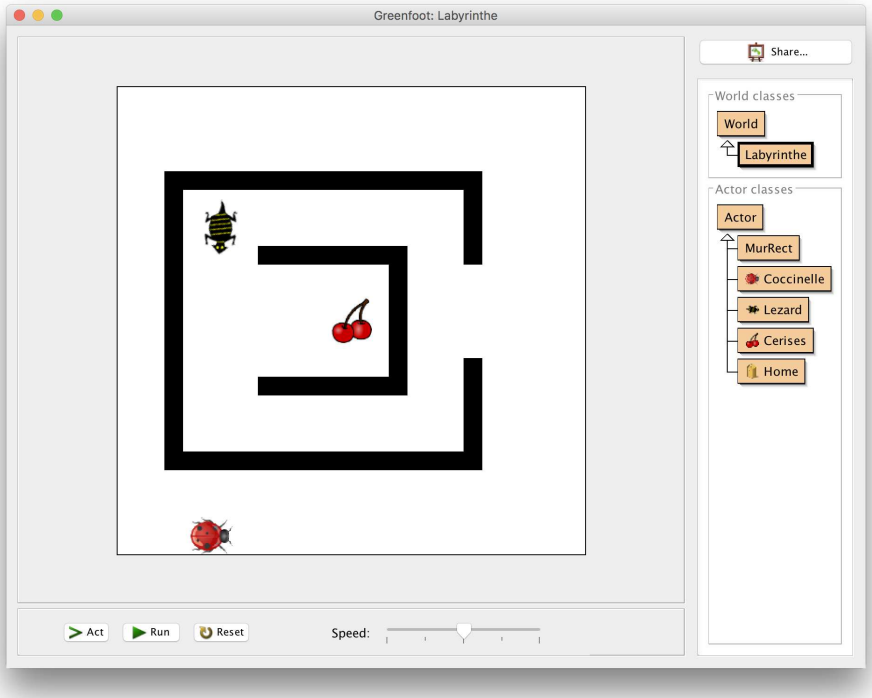
Ce scénario inclut un labyrinthe partiellement terminé, une coccinelle dont les mouvements doivent être ajoutés, et un lézard ennemi que la coccinelle doit éviter. Afin de réussir ce scénario, vous devrez compléter les murs du labyrinthe et ajouter du mouvement et du comportement à votre coccinelle qui lui permettra de se frayer un chemin en toute sécurité à travers le labyrinthe sans passer à travers les murs.



Au lieu de dessiner les murs du labyrinthe directement dans le monde, nous créons un objet bloc rectangulaire de dimensions 20px par 20px (MurRect), et en utilisant des boucles `for`, nous dessinons une série de blocs pour créer les différents murs.

```
// construit un mur horizontal de longueur égale à 17 unités commençant
// au point de coordonnées (60,100).
for (int i = 3; i < 20 ; i++)
{
    rectangle = new MurRect();
    addObject(rectangle, 20*i, 100);
}
```


Greenfoot: Labyrinthe
Share...
World classes
World
Labyrinthe
Actor classes
Actor
MurRect
Coccinelle
Lezard
Cerises
Home
Act
Run
Reset
Speed:

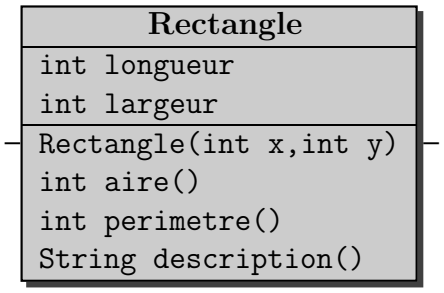

Greenfoot: Labyrinthe
Share...
World classes
World
Labyrinthe
Actor classes
Actor
MurRect
Coccinelle
Lezard
Cerises
Home
Act
Run
Reset
Speed:

2 Exercices sur les objets en java

2.1 Figures géométriques

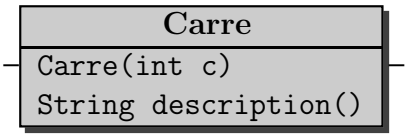
Exercice 2.1

Écrire une classe `Rectangle` avec deux champs `longueur` et `largeur`, avec un constructeur et avec les méthodes `aire`, `perimetre` et `description` (qui donne les caractéristiques du rectangle : longueur, largeur, aire et périmètre). Le constructeur doit stocker dans le champ `longueur` la plus grande valeur des deux paramètres et dans le champ `largeur` la plus petite.



Exercice 2.2

Écrire une sous-classe `Carre` de la classe `Rectangle`. Redéfinir la méthode `description` pour l'adapter à la classe `Carre`.



Exercice 2.3

Créer une méthode `dessineRectangle` de la classe `Rectangle` qui retourne la chaîne de caractères qui permet de dessiner sommairement le rectangle à l'aide d'étoiles.

Créer une classe `TestRectangle` avec une méthode `main` telle que suite au lancement du programme, on obtient la description et le dessin du rectangle ci-dessous.

```
run:
Description du rectangle :
Longueur: 8 unité(s)
Largeur: 6 unité(s)
Aire: 48 unité(s) carrée(s)
Périmètre: 28 unité(s)

* * * * * * * *
*             *
*             *
*             *
*             *
* * * * * * * *

BUILD SUCCESSFUL (total time: 0 seconds)
```

Exercice 2.4

Écrire une classe représentant des cercles. Cette classe contient deux constructeurs :

a) `Cercle()` sans argument qui crée un cercle dont le rayon est égal à 1

b) `Cercle(double r)` avec un argument qui crée un cercle de rayon `r`

Écrire les méthodes suivantes de la classe `Cercle`.

a) pour calculer l'aire

b) pour calculer le périmètre

c) pour obtenir le rayon

d) pour modifier le rayon d'un cercle déjà créé

e) pour donner (sous la forme d'une chaîne de caractères) les caractéristiques (rayon et aire) du cercle

Exercice 2.5

On peut considérer qu'un cylindre est un cercle dont la hauteur est donnée. Construire une classe `Cylindre` qui dérive de la classe `Cercle`.

Créer un constructeur `Cylindre()` sans argument qui crée un cylindre dont le rayon du cercle est égal à 1 et dont la hauteur est égale à 5 et un constructeur `Cylindre(double r, double h)` qui crée un cylindre dont le rayon du cercle est `r` et dont la hauteur est `h`.

Écrire les méthodes qui calculent le volume et l'aire totale d'un cylindre, ainsi que la méthode qui donne les caractéristiques (rayon, aire totale et volume) du cylindre

Formules : *volume* $= \pi \cdot r^2 \cdot h$ et *aire totale* $= 2\,\pi\, r \cdot (r + h)$

Exercice 2.6

Créer une nouvelle classe `TestCylindre` qui contient une méthode `main` telle que suite au lancement du programme, on obtient la description de quelques cercles et cylindres.

2.2 La citerne

Exercice 2.7

Il s'agit de développer une classe simulant une citerne d'eau potable. Développez la classe `Citerne` qui possède les propriétés et fonctionnalités suivantes :

a) une citerne possède un volume maximal de 1000 litres,

b) une citerne a un volume d'eau actuel,

c) on peut ajouter de l'eau à la citerne (attention au volume maximal)

d) on peut retirer de l'eau de la citerne (attention au retrait maximal possible)

e) la citerne renseigne à l'aide d'un indicateur sur son volume actuel,

f) la citerne possède aussi un indicateur affichant le taux de remplissage en pourcents.

Ajouter un constructeur `Citerne()` sans argument qui permet de créer une citerne de 1000 litres vide et une méthode `main` qui simule quelques remplissages et retraits aléatoires successifs.

Exercice 2.8

Sauvez la classe `Citerne` en `CiterneBis`. Cette nouvelle version ne possède plus un volume maximal fixe de 1000 litres, la citerne sera donnée par son rayon et sa hauteur.

Transformer le constructeur `Citerne` en un constructeur `CiterneBis(double rayon, double hauteur)` qui permet de créer des citernes à volume variable selon leurs dimensions extérieures.

N'oubliez pas que le volume maximal ne peut plus être modifié après que la citerne est construite !

2.3 La classe Personne

Exercice 2.9

Créer une classe `Personne` qui permet de définir des objets représentant des personnes. Une personne est décrite par son nom, son prénom, son âge et son sexe. Votre classe doit proposer :

a) Un constructeur par défaut qui ne prend aucun paramètre et qui permet de créer le fameux John Doe (on supposera que ce monsieur a 30 ans).

b) Un constructeur qui prend en paramètre toutes les informations (nom, prénom, âge et sexe) et crée l'objet correspondant convenablement.

c) Un ensemble d'accesseurs (ou getters) qui permettent de récupérer les valeurs des différents attributs de l'objet (ex. une méthode `getName()` qui permet de connaître le nom de la personne et ainsi de suite).

d) Une méthode `sameName(Personne p)` qui prend en paramètre un deuxième objet de type Personne et qui permet de savoir si les deux personnes ont le même nom de famille.

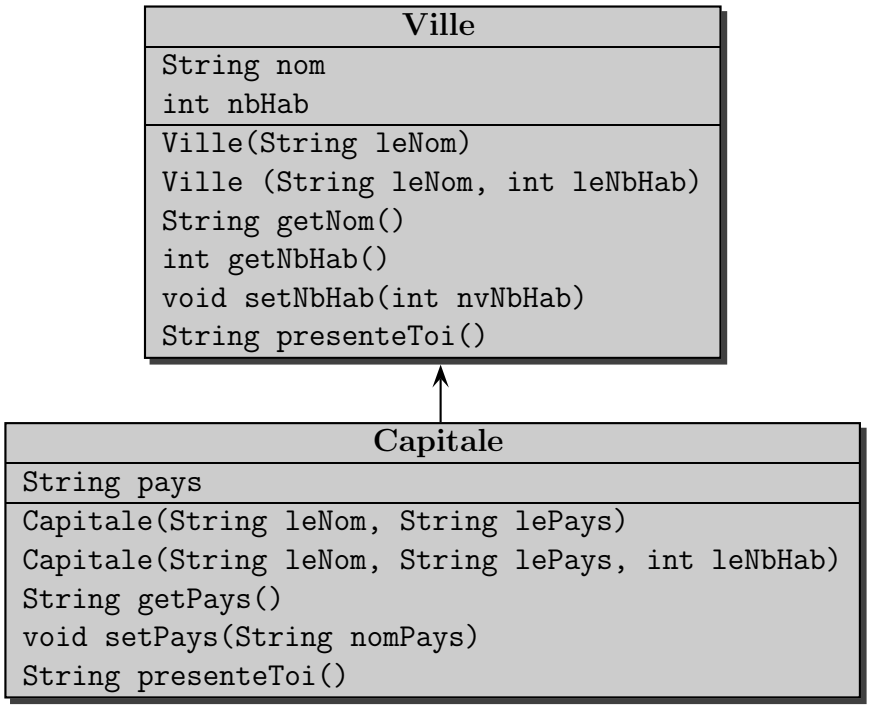
e) Une méthode `oldest(Personne p)` qui compare la personne qui appelle la méthode avec la personne fournie en paramètre et retourne le nom et le prénom de la personne la plus âgée.

Créer une nouvelle classe `TestPersonne` qui contient une méthode `main` où l'on créera le fameux John Doe et la fameuse Jane Doe âgée de 28 ans. Utiliser ensuite la méthode `sameName(Personne p)` pour comparer leurs noms de familles et la méthode `oldest(Personne p)` pour déterminer qui est la personne la plus âgée.

2.4 Ville et Capitale

Exercice 2.10

Écrire les classes `Ville` et `Capitale` selon le diagramme de classe ci-dessous.

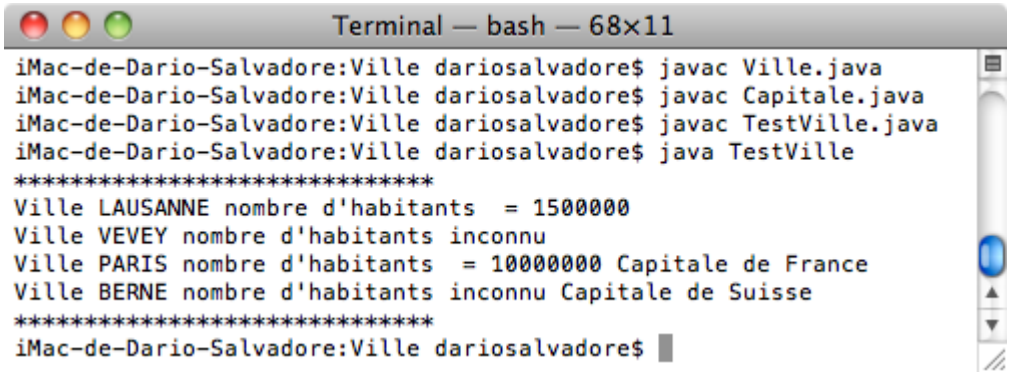


Exercice 2.11

Pour tester ces deux classes, créez la classe `TestVille` suivante :

```
public class TestVille
{
    public static void main(String args[])
    {
        Ville v1 = new Ville("Lausanne", 1500000);
        Ville v2 = new Ville("Vevey");
        Capitale c1 = new Capitale("Paris", "France", 10000000);
        Capitale c2 = new Capitale("Berne", "Suisse");
        System.out.println("******************************");
        System.out.println(v1.presenteToi( ));
        System.out.println(v2.presenteToi( ));
        System.out.println(c1.presenteToi( ));
        System.out.println(c2.presenteToi( ));
        System.out.println("******************************");
    }
}
```

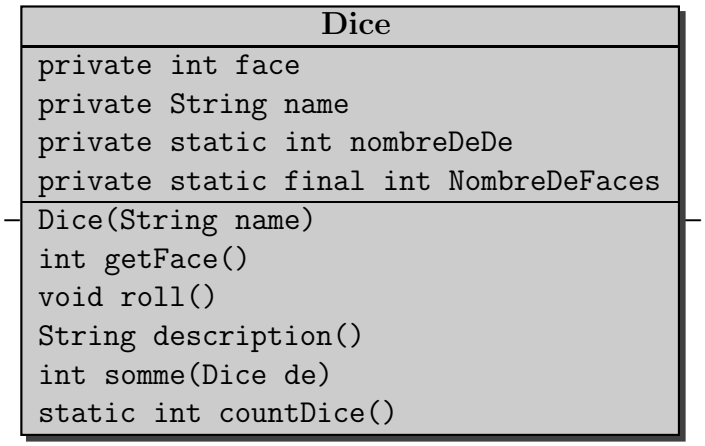
Suite au lancement du programme, on obtiendra la sortie suivante :



2.5 Jouer avec des dés

Exercice 2.12

Voici le diagramme de la classe `Dice`.



Le champ `nombreDeDe` est un champ de classe, il donne le nombre de dés instanciés.

Le champ `NombreDeFaces` est constant et représente ici le nombre de faces d'un dé (6 pour cet exercice).

La méthode `getFace` permet de récupérer la face du dé (accesseur).

La méthode `roll` permet de lancer un dé et ainsi de modifier sa face.

La méthode `description` retourne les informations d'un dé, par exemple : `le dé de1 a la face` 3 ou `le dé n'a pas encore été lancé`.

La méthode `somme` permet d'additionner les faces de deux dés.

La méthode de classe `countDice` retourne le nombre de dés.

A partir de ces informations, créer la classe `Dice`.

Exercice 2.13

Un joueur lance deux dés à six faces et s'intéresse à la somme des deux faces obtenues.

Créer une classe `Joueur` qui permet à d'instancier un joueur avec deux dés et d'obtenir le résultat de la somme du lancer de ses deux dés. Utiliser la classe `Dice` pour simuler le lancer des dés.

Exercice 2.14

Créer une classe `JeuUn` qui permet à deux joueurs de lancer deux dès à six faces. Celui qui obtient le maximum en additionnant les deux faces de ses deux dés a gagné.

Exercice 2.15

Créer une classe `jeuDeux` avec les règles suivantes : un joueur (éventuellement les deux) gagne lorsqu'il obtient un total fixé au départ.

2.6 Devinez un nombre

Exercice 2.16

Écrire un programme `Devinette.java` qui fait deviner un nombre secret compris entre 0 et 100 à quelqu'un. Ce programme commencera par initialiser une variable secret par l'instruction

```
int secret = (int)(Math.random() * 100);
```

ce qui donne une valeur différente à `secret` à chaque exécution du programme. Puis le programme doit faire deviner cette valeur à l'utilisateur en lui demandant d'entrer un nombre et en lui indiquant si celui-ci est supérieur ou inférieur à la valeur secrète jusqu'à ce qu'il trouve le bon nombre.

Par exemple :

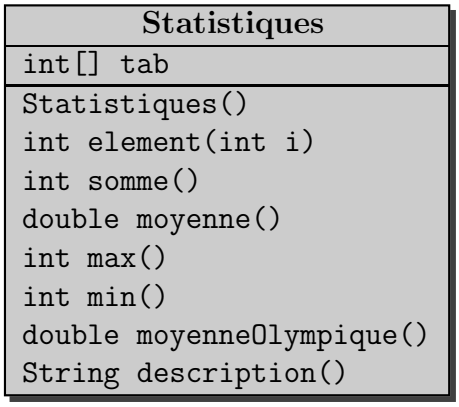
```
Devinez le nombre :
42
Trop grand !
Devinez le nombre :
21
Trop bas !
Devinez le nombre :
23
Trop bas !
Devinez le nombre !
25
Bravo, vous avez gagné !
```

3 Tableaux statiques en Java

3.1 Tableau d'entiers

Exercice 3.1

Voici la classe `Statistiques`.



Le champ `tab` est privé, il contient 10 nombres entiers positifs choisis au hasard entre 0 compris et 100 non compris. La méthode `Math.random()` permettra de générer ces nombres aléatoires.

La méthode `element(int i)` renvoie le $i^{\text{ème}}$ élément de `tab`.

La méthode `moyenneOlympique` est la moyenne des éléments de `tab` obtenue en lui retirant son plus grand et et son plus petit élément.

Les autres méthodes sont explicites.

a) Créer la classe `Statistiques`, puis la tester dans une classe `TestStatistiques`.

b) Ajouter à la classe `Statistiques` une méthode statique dont la signature est
`public static void echange(int[] tableau, int i, int j)`
qui permute dans `tableau` les deux coefficients d'indice `i` et `j`.

c) Ajouter à la classe `Statistiques` une méthode `int[] elementsPairs()` qui renvoie un tableau composé des éléments pairs de `tab`.

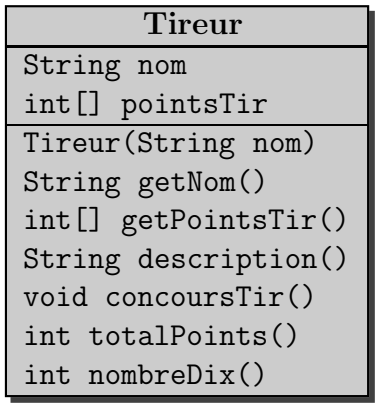
d) Ajouter à la classe `Statistiques` une méthode `int[] elementsImpairs()` qui renvoie un tableau composé des éléments impairs de `tab`.

Exercice 3.2

Ajouter à la classe `Statistiques` une méthode `int[] tri()` qui renvoie le tableau formé des éléments de `tab` triés par ordre croissant.

Exercice 3.3

Créer un projet NetBeans dont le nom est `tir` avec une classe `Tireur` à créer selon le diagramme fourni ci-dessous et une classe `ConcoursTir` qui contient la classe `main`.



— Le champ `pointsTir` est un tableau stockant la liste des cinq tirs du tireur sur une cible à 10 points.

— Le constructeur `Tireur(String nom)` permet de créer un tireur avec un nom. Ce constructeur initialise ensuite le champ `pointsTir` en lui affectant un tableau de longueur cinq. Les valeurs contenues dans `pointsTir` sont enfin toutes initialisées explicitement à -1 (ce qui signifie que les cinq tirs n'ont pas encore été effectués).

a) On veillera à respecter les points ci-dessous :

— La méthode `description` renvoie une chaîne de caractères qui contient le nom du tireur, suivi, à la ligne suivante, par la suite des cinq résultats du tir. Par exemple

Tireur : Jean

Résultats : 8, 2, 9, 10, 0

Si le tireur n'a pas encore effectué son tir, la méthode `description` indique que le tir n'a pas encore eu lieu. Par exemple

Tireur : Jean

Résultats : en attente

— La méthode `concoursTir` simule cinq tirs du joueur : pour chacun des cinq tirs, elle fixe au hasard un nombre entier de points entre 0 et 10 (compris) et le stocke dans `pointsTir`.

— La méthode `totalPoints` retourne le total des points obtenu par le tireur.

— La méthode `nombreDix` retourne le nombre de 10 obtenu(s) par le tireur.

b) Dans la méthode `main` de la classe `ConcoursTir`, créez le joueur dont le nom est *Jean*. Prouver ensuite que *Jean* n'a pas encore tiré.

c) Dans la méthode `main` de la classe `ConcoursTir`, faire effectuer cinq tirs à *Jean* et afficher à l'écran ses résultats. Afficher ensuite le total obtenu et le nombre de 10 obtenu(s).

d) Dans la classe `Tireur`, créer une méthode dont la signature est

`public static String MeilleurTireur(Tireur t1, Tireur t2)`

qui retourne une chaîne de caractères indiquant entre les tireurs *t1* et *t2* qui a obtenu le meilleur total de points (ne pas oublier de traiter l'égalité de totaux de points).

e) Dans la méthode `main` de la classe `ConcoursTir`, créer le joueur dont le nom est *Pierre*. Faire effectuer cinq tirs à *Pierre* et afficher à l'écran ses résultats et le total de ses points.

f) Utiliser la méthode `MeilleurTireur` pour comparer à l'écran les résultats de *Jean* et *Pierre*.

3.2 Classe Equipe

Exercice 3.4

Créer un projet NetBeans `EquipeBasket`. La classe `Equipe` est donnée par le diagramme suivant :

Equipe

`String nom`
`String[] joueurs`
`int[] pointsMatch`

`Equipe(String nom, String[] joueurs)`
`String getNom()`
`String[] getJoueurs()`
`int[] getPointsMatch()`
`String description()`
`void joueMatch()`
`int totalPoints()`
`int maxPoints()`
`String meilleursMarqueurs()`

— Le champ `joueurs` est un tableau stockant la liste des noms des joueurs de l'équipe.

— Le champ `pointsMatch` mémorise, dans l'ordre, les points de chaque joueur lors d'un match donné.

— Le constructeur `Equipe(String nom, String[] joueurs)` permet de créer une équipe avec un nom et une liste de joueurs prédéfinis. Ce constructeur initialise ensuite le champ `pointsMatch` en lui affectant un tableau de la même longueur que le celle du tableau `joueurs` donné en paramètre. Les valeurs contenues dans `pointsMatch` sont enfin toutes initialisées explicitement à 0.

a) Créer les méthodes de la classe `Equipe` en tenant compte des points ci-dessous :

— La méthode `description` renvoie une chaîne de caractères qui contient le nom de l'équipe, suivi, à la ligne suivante, par le contingent. Par exemple

Equipe : Chicago Bulls
Effectif : Jordan, Sefolosha, Pippen, Rodman, Gilmore

— La méthode `joueMatch` simule un match de l'équipe : pour chaque joueur, elle fixe au hasard un nombre entier de points entre 0 et 30 (compris) et le stocke dans `pointsMatch`.

— La méthode `totalPoints` retourne le total des points marqués par les joueurs de l'équipe.

— La méthode `maxPoints` retourne le nombre maximum de points marqués par un des joueurs de l'équipe.

— La méthode `meilleursMarqueurs` renvoie une chaîne de caractères qui contient le nom du (ou des) meilleur(s) marqueur(s) de l'équipe pendant le match. Par exemple

Meilleur(s) marqueur(s) de Chicago Bulls avec 26 points : Gilmore, Pippen

b) Dans la méthode `main` d'une classe `TestBasket`, créez l'équipe *bulls* dont le nom est *Chicago Bulls* et dont les noms des joueurs sont les suivants :
`Jordan, Sefolosha, Pippen, Rodman, Gilmore`.
Afficher ensuite à l'écran le nom de cette équipe et son contingent.

c) Dans la méthode `main` de la classe `TestBasket`, créez l'équipe *lakers* dont le nom est *Los Angeles Lakers* et dont les noms des joueurs sont les suivants :
`Bryant, Johnson, Abdul Jabbar, O'Neal, Scott`.
Afficher ensuite à l'écran le nom de cette équipe et son contingent.

Exercice 3.5

La classe `Match` est donnée par le diagramme suivant :

Match
`Equipe equipe1` `Equipe equipe2`

`Match(Equipe equipe1, Equipe equipe2)`
`Equipe getEquipe1()`
`Equipe getEquipe2()`
`String gagnantMatch()`

Les champs `equipe1` et `equipe2` stockent les deux objets de type `Equipe` qui font un match.

a) Créer le constructeur et les méthodes de la classe `Match` en tenant compte des points ci-dessous :

— Le constructeur `Match(Equipe equipe1, Equipe equipe2)` permet de créer un match entre les équipes `equipe1` et `equipe2`. Il lance également pour chaque équipe la méthode `joueMatch`.

— La méthode `gagnantMatch` retourne le résultat du match. Par Exemple :

Match Chicago Bulls - Los Angeles Lakers
Chicago Bulls gagne 106 - 87

Attention à ne pas oublier le cas d'un match nul.

b) Ajouter dans la méthode `main` de la classe `TestBasket` le code permettant de faire jouer un match entre *Chicago Bulls* et *Los Angeles Lakers* et afficher à l'écran le résultat du match.

c) Afficher ensuite à l'écran le(s) meilleur(s) marqueur(s) des deux équipes.

3.3 Triangle de Pascal

Exercice 3.6

Nous allons compléter la classe `Pascal`

Pascal
`int[][] tableau`

`Pascal(int n)`
`int[][] getTableau()`
`int[] getLigne(int n)`
`String affiche()`
`String afficheSomme()`

Le champ `tableau` est privé. C'est un tableau de tableaux ; les éléments sont les lignes successives du tableau de Pascal sous forme de tableaux.

Le constructeur `Pascal(int n)` construit un tableau de Pascal de niveau `n`. Par exemple le tableau de Pascal de niveau 3 est le suivant :

1

1-1

1-2-1

1-3-3-1.

La méthode `getLigne(int n)` retourne la `n`-ième ligne du tableau de Pascal (1 est la ligne 0, 1-1 est la ligne 1, 1-2-1 est la ligne 2 et ainsi de suite).

La méthode `affiche()` retourne une chaîne de caractères permettant d'afficher le tableau de Pascal dans la fenêtre du terminal.

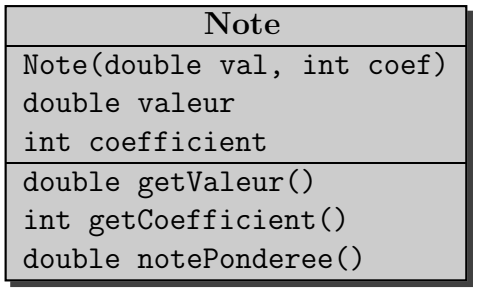
La méthode `afficheSomme()` retourne une chaîne de caractères permettant d'afficher la suite des sommes des lignes du tableau de Pascal.

4 Listes dynamiques en Java

4.1 Les classes Note et Branche

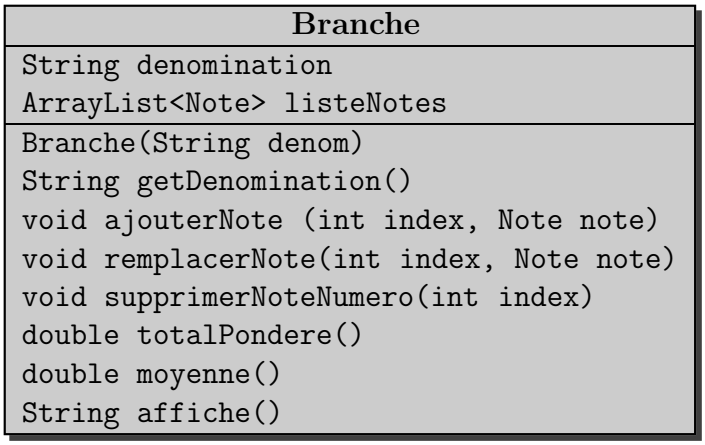
Exercice 4.1

Dans un projet NetBeans appelé `moyennes`, créer une classe correspondant au diagramme ci-dessous:



Cette classe dispose d'un champ privé de type `double` qui permet de stocker la note et d'un champ privé de type `int` pour tenir compte du coefficient de la note. On peut consulter la note et son coefficient, mais pas les modifier après construction. On peut également faire calculer la note pondérée (produit de la valeur par le coefficient).

Créer ensuite une deuxième classe donnée par le second diagramme :



Cette classe dispose d'un champ privé de type `String` pour nommer la branche et d'un champ privé de type `ArrayList<Note>` qui permet de stocker les notes au fur et à mesure.

a) Dans la méthode `main` de la classe `Moyennes`, créer un objet de type `Branche` qui s'appelle `francais`, par exemple. À l'aide d'une boucle, remplir ensuite cet objet avec une quinzaine de notes calculées aléatoirement. Les dix premières notes comptent simple et les cinq dernières comptent double.

b) Dans la javadoc, passer en revue les méthodes prédéfinies de la classe `ArrayList`.

c) Faire afficher toutes les notes de français.

d) Remplacer la première note de français insuffisante par un 4 (attention à conserver la pondération).

e) Ajouter un 6 de coefficient 1 au début de la liste des notes de français.

f) Faire afficher le total pondéré des notes de français et la moyenne de cette branche à la console.

g) Créer une classe `EnsembleDesNotes` qui contient toutes les notes d'un élève dans les branches suivantes : français, maths, allemand, histoire, géographie, physique, chimie et biologie. les noms des branches sont stockés dans un tableau statique et le constructeur de cette classe est chargé de créer un objet de type `Branche` pour chacun de ces noms. On veut pouvoir consulter les différentes moyennes et gérer les notes de chaque branche.

4.2 La classe Personne

Exercice 4.2

Créer une `classe Personne`.

Voici les fonctionnalités de `Personne` :

— des variables d'instances, privées, sont : String nom, prenom, sexe ; int anneNaissance

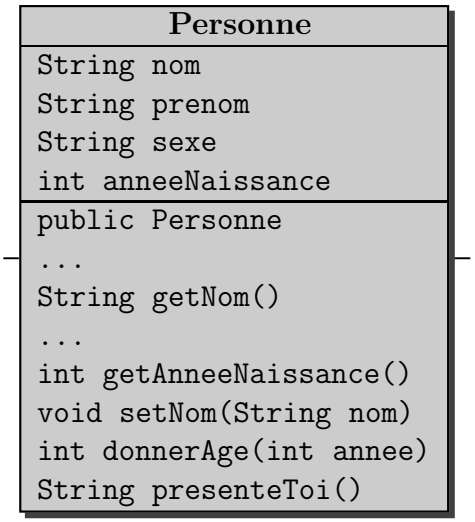
— deux constructeurs :

```
public Personne (String n, String p, String s){..}
public Personne (String n, String p, String s, int a){..}
```

— des méthodes d'instances publiques pour accéder aux variables privées `getNom(), ..., getAnneeNaissance(),setNom()`

— une méthode `donnerAge(int annee)` qui retourne l'âge de la personne pendant l'année donnée en paramètre

— une méthode `presenteToi()` pour afficher un texte comme :
`Mme Durant Jeanne est née en 1972.`



Exercice 4.3

Modifier la classe `Personne` pour qu'elle permette de décrire une personne avec les informations données dans la phrase suivante :

`M. Holly Pierre est né en 1965, il est marié.`

Exercice 4.4

Écrire dans une classe `TestPersonne` un programme qui crée 3 instances de `Personne` et affiche les informations les concernant.

Exercice 4.5

Ajouter à la classe `Personne` un attribut conjoint et examiner les conséquences que cela peut avoir sur l'ensemble du code.

Exercice 4.6

Ajouter une méthode `marier(Personne p)` qui permet de marier une personne à une autre. Modifier la méthode `presenteToi()` de façon que le nom des époux soient modifiés comme suit :

Pour simplifier : quand une femme se marie son nom devient : "[nom de l'époux] née [nom de jeune fille]", par exemple : si Mlle Jeanne Durant se marie avec M. Jean Dupond, sa présentation deviendra "Mme Dupond Jeanne née Durant en 1971, est mariée avec Dupond Jean".

Pour un homme marié, on ajoutera le nom de l'épouse, par exemple : si M. Jean Dupond se marie avec Mlle Jeanne Durant, sa présentation deviendra "M. Dupond Jean né en 1971 est mariée avec Durant Jeanne".

Si le conjoint n'est pas connu, on conservera la présentation précédente.

Attention : cet exercice ne prétend pas représenter l'état actuel de la loi !

Exercice 4.7

Créer une classe `GroupePersonnes` qui contient un groupe de personnes. Le groupe possède un nom et un champ privé de type `ArrayList<Personne>` qui permet de stocker au fur et à mesure les membres qui rejoignent le groupe.

Une méthode `ajouterPersonne(Personne p)` permet d'ajouter `p` au groupe de personnes.

Une méthode `presenteGroupe()` retourne une chaîne de caractères qui donne le nom du groupe et la description de ses membres. Créer un groupe dans la classe `TestPersonne` et le faire se présenter.

5 Publication web

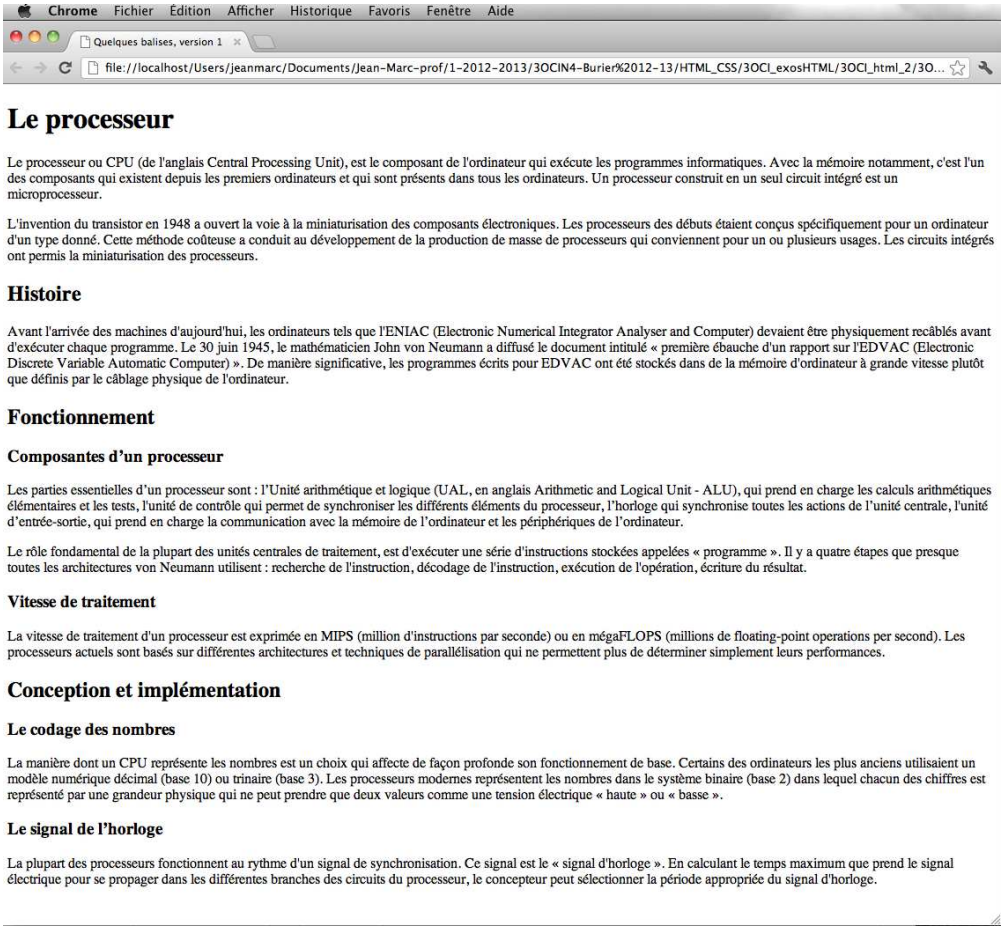
5.1 HTML 5

Exercice 5.1

Enregistrer le code source minimal d'une page HTML. Ouvrir le fichier à la fois dans un éditeur de texte et dans un navigateur. Observer l'effet de la modification du contenu de la balise `<title></title>` en actualisant la page ouverte dans le navigateur. Faire cette manipulation dans plusieurs navigateurs.

Exercice 5.2

À partir d'un fichier HTML minimal et du document CPU.txt, créer la page ci-dessous :

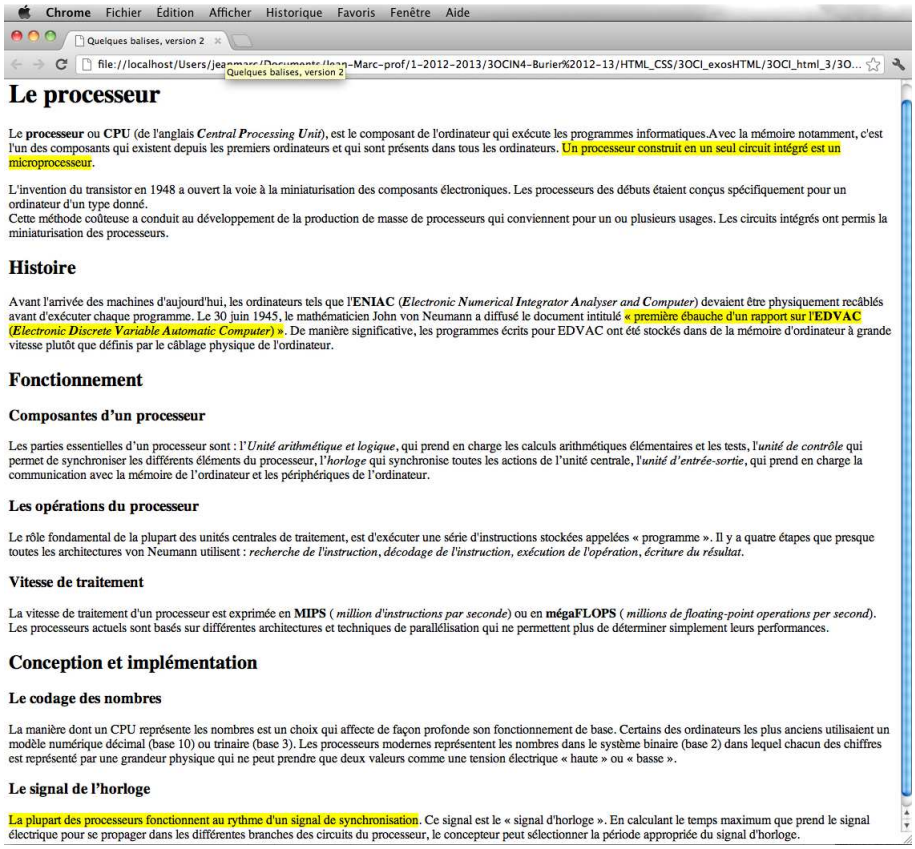


Il convient d'utiliser les balises suivantes :

`<h1></h1>`, `<h2></h2>`, `<h3></h3>` et `<p></p>`.

Exercice 5.3

Modifier le code source de la solution de l'exercice précédent pour obtenir la page ci-dessous :

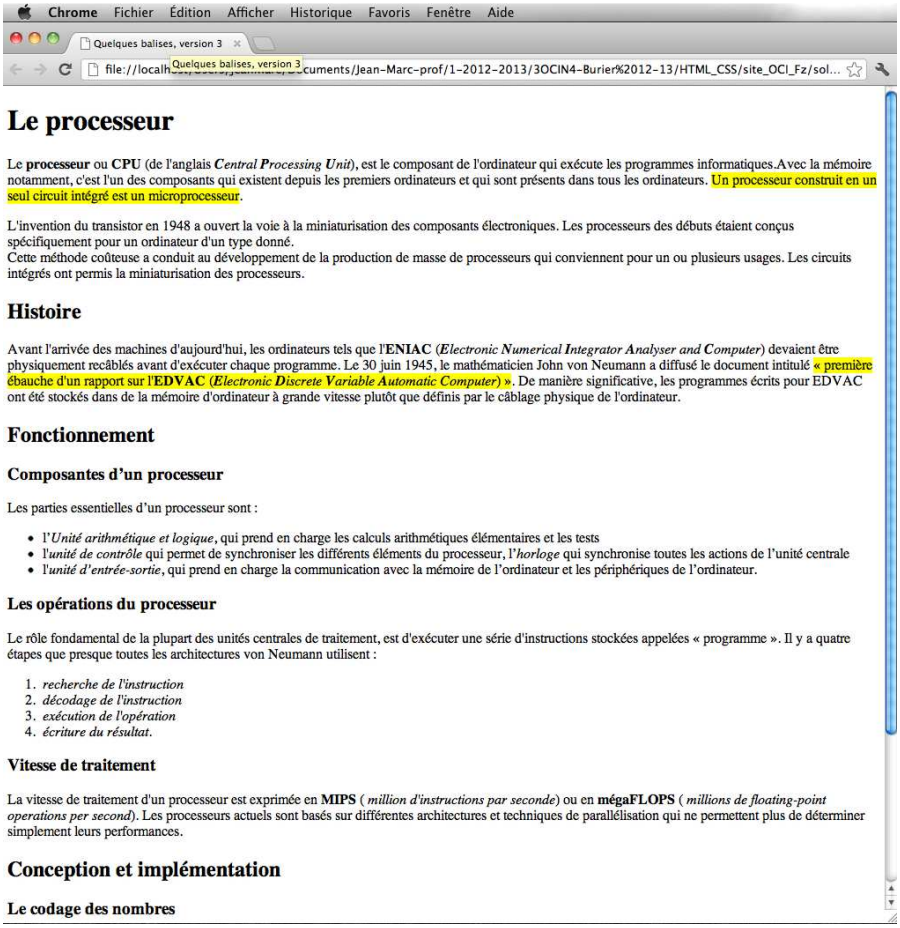


Il convient d'utiliser les balises suivantes :

`</br>`, `<em></em>`, `<strong></strong>` et `<mark></mark>`.

Exercice 5.4

Modifier le code source de la solution de l'exercice précédent pour obtenir la page ci-dessous :



Il convient d'utiliser les balises `<ul></ul>`, `<ol></ol>` et `<li></li>`.

Exercice 5.5

Créer un dossier dont le nom est `site_OCI_nomDeFamille`, en remplaçant `nomDeFamille` par le vôtre. Dans ce dossier, ajouter votre fichier `index.html` et créer un sous-dossier `solutions` contenant le fichier `html` dans lequel se trouve le code source de la solution de l'exercice 4. Compléter le fichier `index.html` en ajoutant un lien permettant de consulter la solution de l'exercice 4. Un lien hypertext "site de mon école" doit renvoyer à la page d'accueil du gymnase de Burier (s'inspirer de l'exemple ci-dessous).

Il convient d'utiliser la balise `<a href="..."></a>`. Rendez ensuite votre solution atteignable via votre page du site www.burier.ch.

Exercice 5.6

Créer le fichier texte `actionHumaine.html` à partir d'un document `HTML` de base. À l'aide des balises adéquates et d'un certain nombres de copier-coller, mettre en forme le texte du fichier `actionHumaine.txt` dans le fichier `actionHumaine.html` conformément au fichier `actionHumaine.pdf`.

a) Ajouter à cette page internet un lien sur la page de Wikipédia consacrée à Ludwig von Mises et un lien sur le site des Presses Universitaires de France.

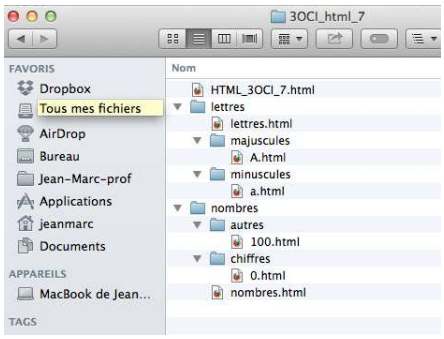
b) Établir les liens qui permettent, d'une part, d'accéder aux deux notes de bas de document et, d'autre part, de remonter à l'emplacement de ces notes dans le texte.

c) En haut du document, créer une table des matières permettant d'accéder au début de chaque paragraphe. Faire en sorte de pouvoir remonter à l'emplacement correspondant dans la table des matières en cliquant sur l'un quelconque des titres de paragraphe. Après les notes, un lien hypertext "Retour vers la table des matières" permet de remonter au début de la Table des matières.

Exercice 5.7

En consultant la page web `3OCI_7.html` et en cliquant les liens auxquels on peut accéder, deviner la structure du répertoire dans lequel se trouve ce fichier. On s'intéresse à la structure des dossiers adjacents au fichier `3OCI_7.html` et à tous les éventuels sous-dossiers.

Créer un « site » selon l'arborescence ci-dessous, ayant comme propriété que l'on peut accéder à toutes les pages du site depuis une « feuille » quelconque de l'arborescence. On peut de plus parcourir l'arbre de la racine aux feuilles.



Exercice 5.8

Créer une page web basique qui s'appelle `3OCI_8.html`. On trouve sur cette page une liste non-ordonnée de liens qui permet

a) d'accéder directement aux notes du premier paragraphe du livre de Ludwig von Mises, sur la page de l'exercice 6 ;

b) d'accéder au site *http ://burier.ch* en affichant le texte *Rien que du bon* dans une infobulle ;

c) d'ouvrir une nouvelle fenêtre qui s'ouvre sur la première page du tutoriel `Java` du site du Zéro ;

d) d'envoyer un courriel au rédacteur de l'exercice ;

e) de télécharger un fichier `zip` qui contient le code source des fichiers `html` solutions des exercices 4 et 6.

Exercice 5.9

Créer la page `New_York.html` en y insérant l'image `Flat-Iron-Building.jpeg`.

Les images se trouvent dans le dossier `imagesExosHtml`. Elles sont à placer dans un dossier nommé `images_NY`.

Ajouter l'*infobulle* avec le texte `Flat Iron Building`.

Exercice 5.10

Compléter la page `New_York.html` en insérant les trois images

`Manhattan-Skyline.jpeg`, `Statue-of-Liberty.jpeg` et `Times-Square.jpeg`.

Pour chaque image créer une *infobulle*.

Exercice 5.11

On peut à l'aide du menu *outils* du logiciel **Aperçu** créer pour chaque image une miniature correspondante en dupliquant l'image et en ajustant sa taille. Créer des miniatures des images des quatre building et les placer dans un dossier `miniatures_NY` contenu dans le dossier `images_NY`.
Créer la page `New_York_Mini.html` en utilisant cette fois-ci les miniatures. Les miniatures sont cliquables et renvoient aux images en taille originale. Les images doivent s'ouvrir dans une nouvelle fenêtre.

Exercice 5.12

Utiliser les balises `<figure>` et `<figcaption>` pour obtenir la nouvelle page html.

Exercice 5.13

A l'aide du logiciel **Aperçu** transformer l'image `i-love-new-york.png` au format 32 x 32 pixels et la nommer `favicon.png`. Ensuite, insérer à l'intérieur de la balise en-tête `<head></head>` l'instruction

`<link rel="icon" type="image/png" href="images_NY/favicon.png" />`.

Observer l'effet de cette commande dans différents navigateurs.

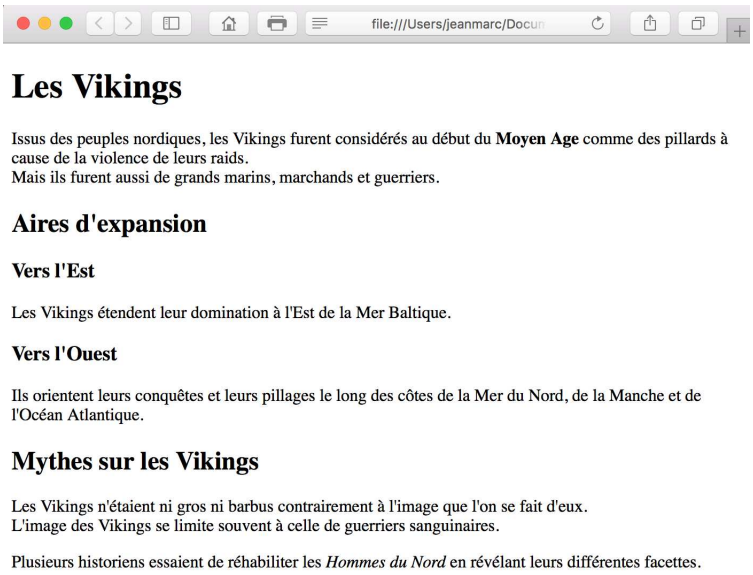
Exercice 5.14

Créer la page `mon_cv.html` sur laquelle figure votre *curriculum vitae* et y insérer votre photo.

5.2 CSS 3

Exercice 5.15

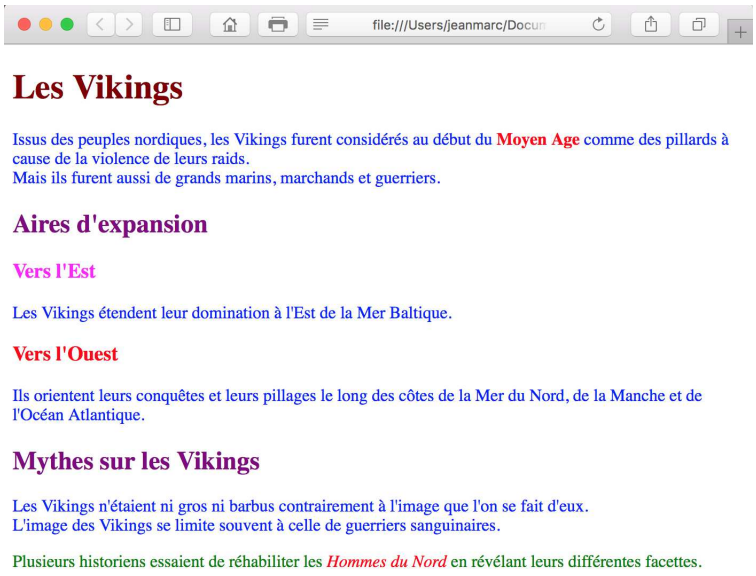
A l'aide du document Viking.txt, réaliser le document ci-dessous en le sauvant sous le nom Viking1.html.



Créer ensuite une feuille de style externe en la sauvant dans un dossier "styles" sous VikingStyle.css. L'objectif est d'obtenir le document représenté en page suivante. Associer ensuite la feuille de style dans le document Viking1.html et le sauver sous Viking2.html.

Respecter les éléments suivants :

1) l'attribut `color` pour la couleur du texte utilise les couleurs suivantes : `maroon`, `purple`, `fuchsia`, `blue`, `red`, `green`.
2) utiliser l'attribut `id` afin que le dernier paragraphe apparaisse dans la couleur voulue.
3) utiliser l'attribut `class` afin que les deux titres de niveau 3 apparaissent dans les couleurs voulues.



Sauver le document Viking2.html sous Viking3.html.

Effectuer ensuite les modifications suivantes sans supprimer la balise `<link>` (qui fait référence à la feuille de style VikingStyle.css) :

— Ajouter une feuille de style interne (donc une balise `<style>` dans l'en-tête `<head>`) qui fixe la couleur du texte d'une balise `<p>` à `gray` et d'une balise `<strong>` à `black`.

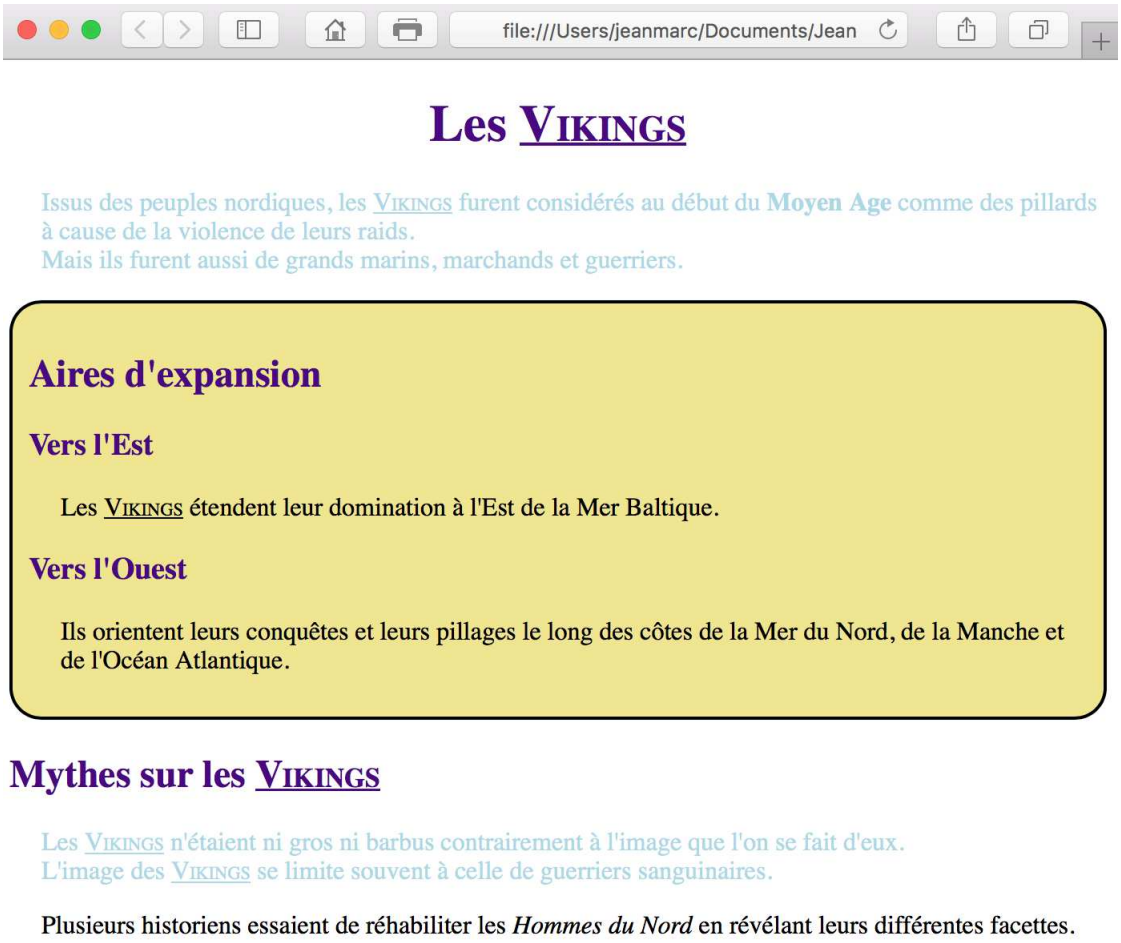
— Ajouter directement sur la balise de l'avant dernier paragraphe un style afin que la couleur du texte apparaisse en `lime` (on parle alors de style en ligne).

— Ajouter un style en ligne de telle sorte que la couleur du texte du deuxième titre de niveau trois soit `teal`.

Quelle est la hiérarchie entre les feuilles de style externe, interne et les styles en ligne ? Pourquoi le dernier paragraphe ne change-t-il pas de couleur ?

Exercice 5.16

Sauvez Viking1.html sous Viking4.html et ajouter une feuille de style interne de façon à obtenir le résultat suivant :



L'arrière-plan du cadre, créé à l'aide d'une balise universelle `<div>`, est de couleur khaki, les titres de couleur indigo, les deux paragraphes mis en évidence de couleur lightblue.

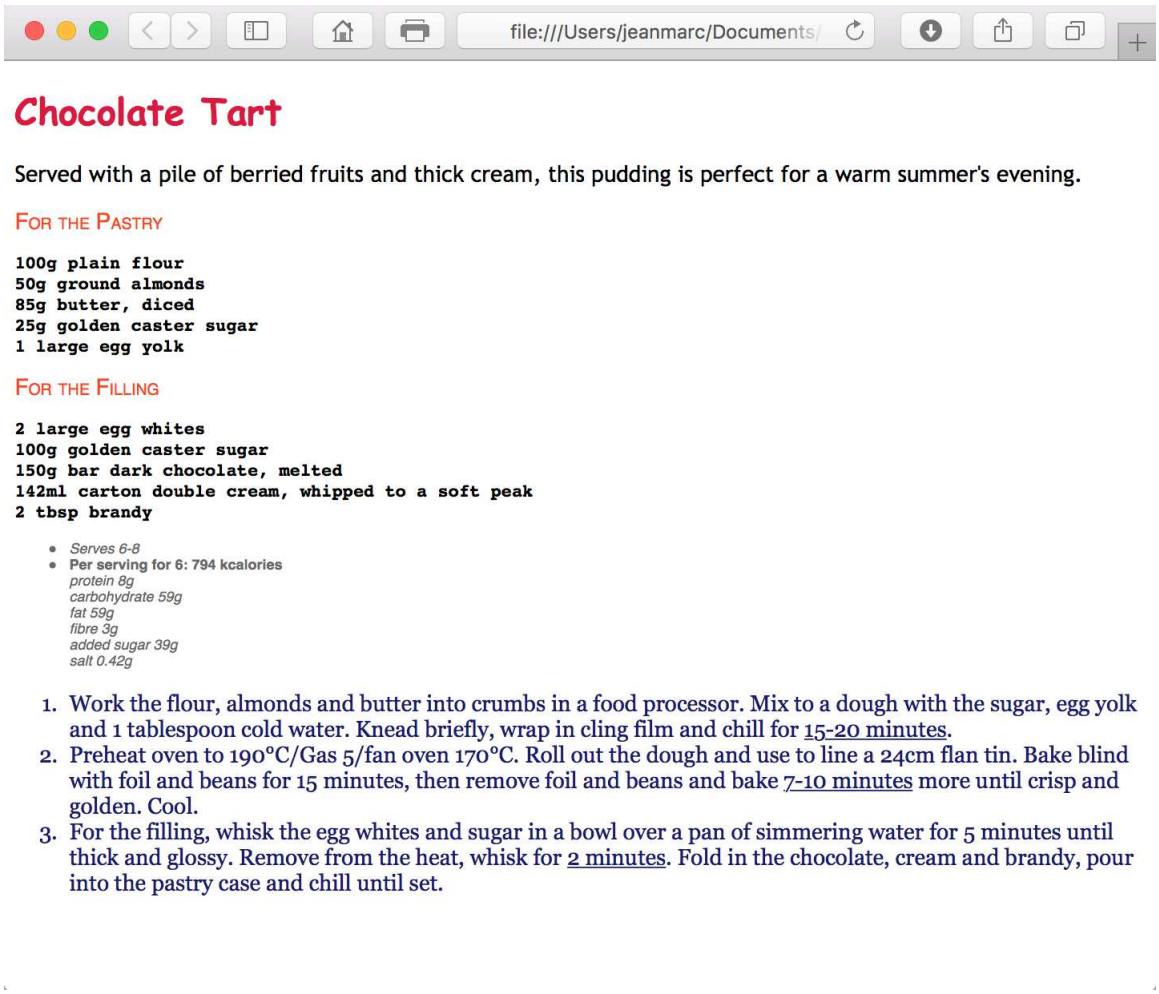
Les polices sont "Comic sans MS" (sinon sans-serif) pour les titres et "Trebuchet MS" (sinon serif) pour les paragraphes.

Les coins arrondis s'obtiennent avec l'attribut `border-radius`, l'espacement entre le cadre et le texte avec l'attribut `padding`, les marges gauches pour les paragraphes avec l'attribut `margin-left`.

Toutes les occurrences du mot *Vikings*, placées dans une balise universelle `<span>`, sont soulignées (avec l'attribut `text-decoration`) et notées en petites capitales (avec l'attribut `font-variant`).

Exercice 5.17

En utilisant le texte `Chocolate_Tart.txt` créer la page suivante en la sauvant sous `recette1.html` et en utilisant une feuille de style externe nommée `recetteStyle1.css` :

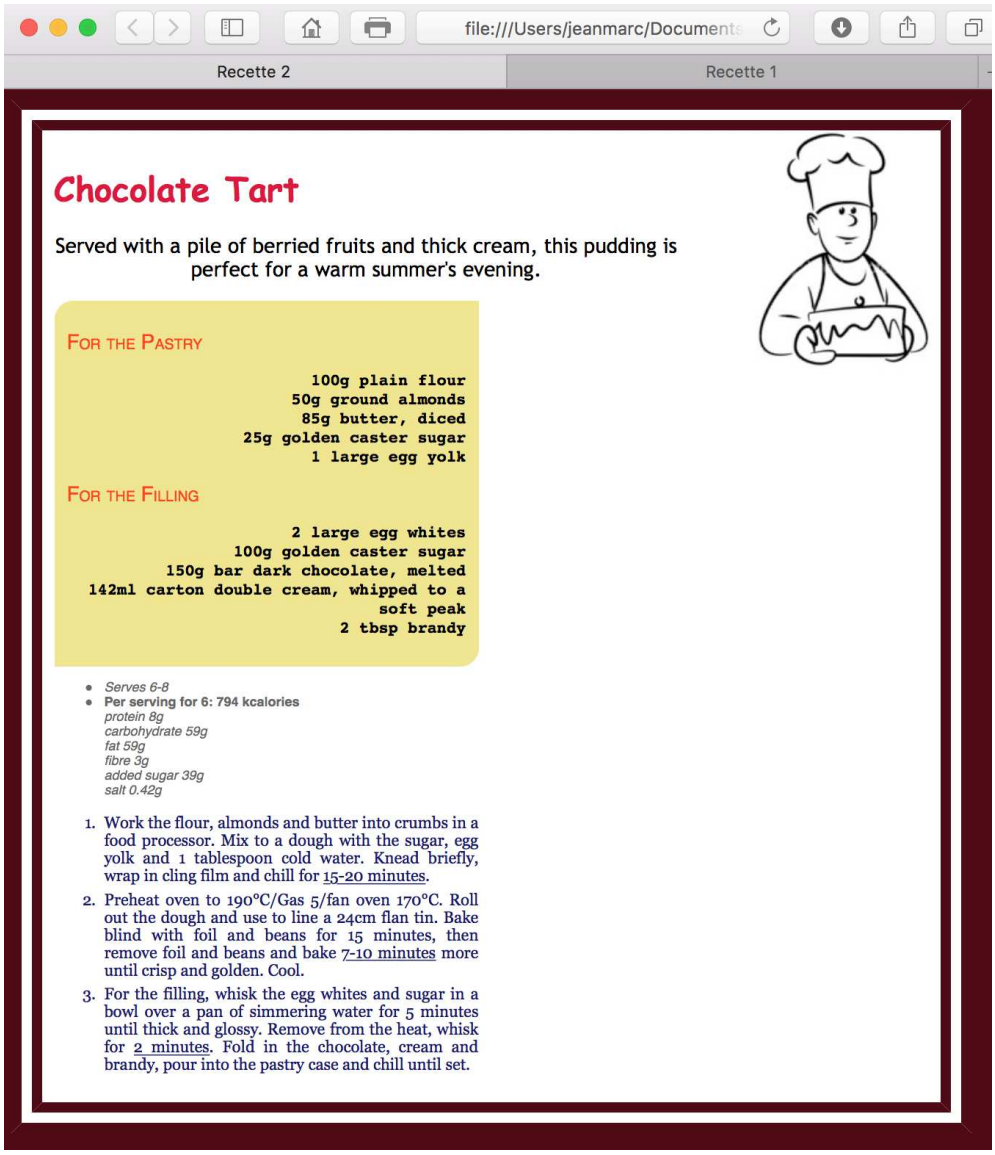


Les couleurs utilisées sont `crimson` pour le titre, `orangered` pour les sous-titres, `dimgrey` pour les indications nutritives et `midnightblue` pour la marche à suivre. Les polices sont `"Comic Sans MS"` (sinon `cursive`) pour le titre, `"Trebuchet MS"` (sinon `sans-serif`) pour l'introduction, `Verdana` (sinon `sans-serif`) pour les sous-titres et les indications nutritives, `"Courier New"` (sinon `monospace`) pour les ingrédients et `Georgia` (sinon `serif`) pour la marche à suivre.

Les tailles des polices sont 20px pour le titre, 12px pour les sous-titres et l'introduction, 10px pour les ingrédients et la marche à suivre, 8px pour les indications nutritives. Les ingrédients sont en gras, alors que les sous-titres ne doivent pas l'être. Ces derniers doivent par ailleurs être en petites capitales. Les indications de temps sont soulignées, on prendra garde à mettre en italique les informations nutritives qui ne sont pas en gras.

Exercice 5.18

Sauvez `recette1.html` et `receteStyle1.css` sous `recette2.html` et `receteStyle2.css`. Modifier les deux nouveaux documents de telle sorte à créer la page suivante :



L'introduction a une largeur de 500px ; la liste des ingrédients et la marche à suivre ont toutes deux une largeur de 300px. L'ensemble de la recette est contenue dans une boîte principale de largeur 700px.

Le texte de l'introduction est centré, les ingrédients sont alignés à droite. Le texte de la marche à suivre est justifié, Il y a un espace vertical de 5px entre les différentes étapes de la marche à suivre.

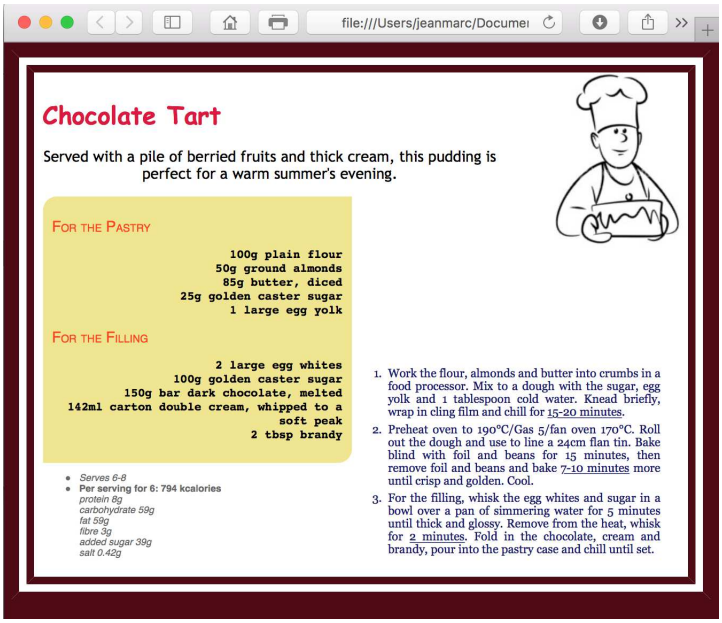
Le petit cadre avec les ingrédients a une marge intérieure de 10px.

Les couleurs de fond sont `khaki` pour le cadre et `rgb(80,10,20)` pour la bordure et la couleur de fond du body.

Utiliser la propriété `overflow` afin que le texte de la boite principale ne soit pas coupé. Ne pas oublier de placer l'image de fond.

Exercice 5.19

Compléter les fichiers `recette2.html` et `recetteStyle2.css` afin d'obtenir le résultat suivant :



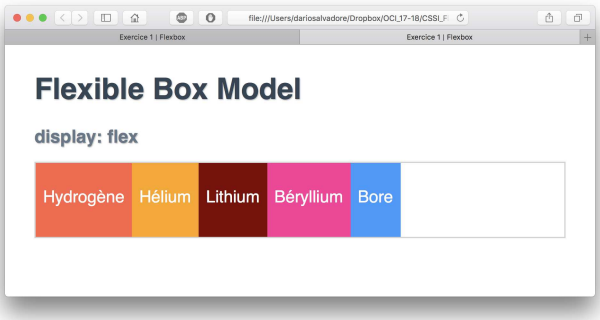
A l'aide de boîtes universelles `<div>` et de la mise en page avec `Flexbox`, disposer la marche à suivre à droite du groupe formé par les ingrédients et les indications nutritives.

5.3 Positionnement avec Flexbox

Exercice 5.20

Dans le fichier `ex1_flexbox.html`, créer un lien vers un fichier `style.css`.

Nous voulons obtenir le résultat suivant :



Dans ce fichier CSS, voici les instructions à ajouter :

a) L'élément `body` a comme propriétés :

   i) Une marge extérieure égale à zéro et une marge intérieure de 10px à gauche et à droite et de 40px en haut et en bas

   ii) La taille des polices (`helvetica, arial, serif`) est de `1.4em`

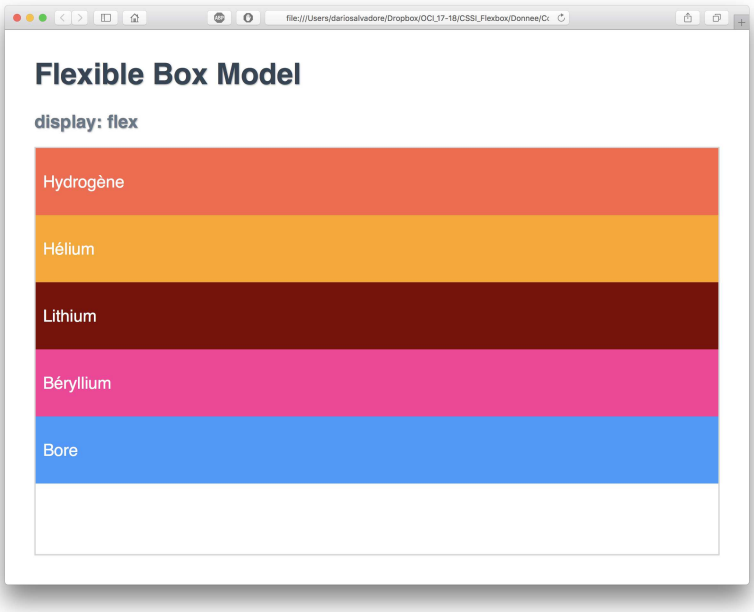
b) La taille de la police de l'élément `h1` est de `1.8em`

c) La taille de la police de l'élément `h2` est de `1.1em`

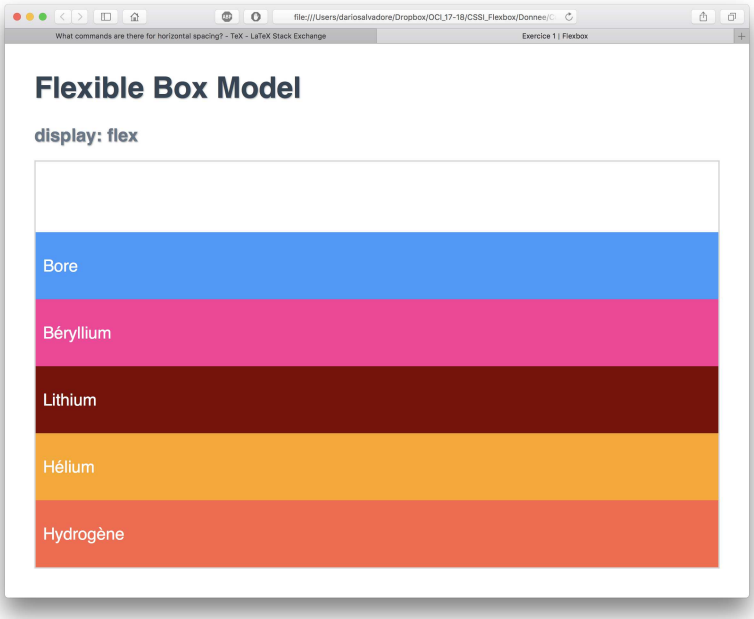
d) Les couleurs des éléments `item` sont donnés dans l'ordre par `tomato, orange, #800000, #FF3399` et `#3399FF`

Puis créer pour chaque cas une copie de votre dossier qui contient l'exercice 1 et modifier le fichier css pour obtenir les pages présentées ci-dessous.

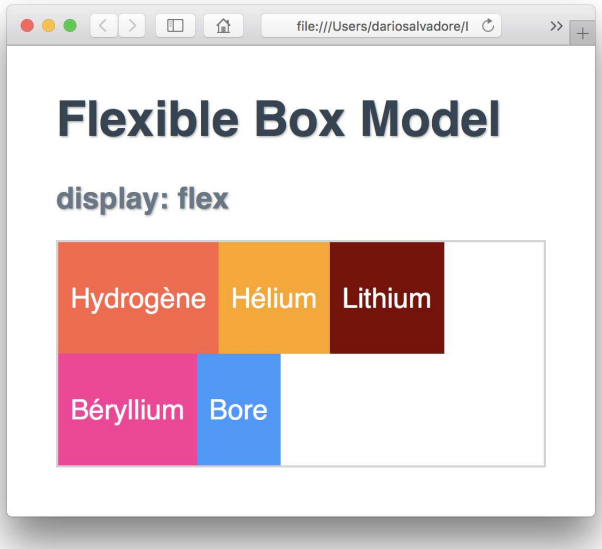
La hauteur de l'élément `container` est de 550px.



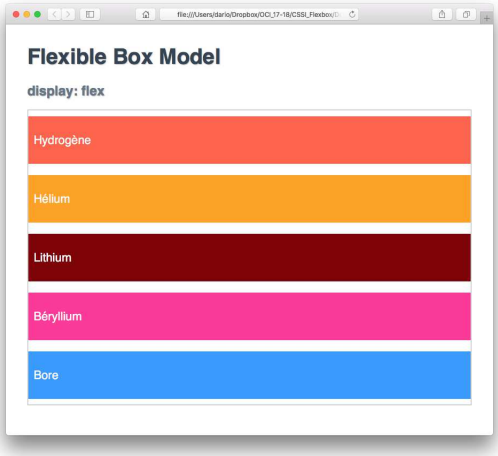
Comment obtient-on la page présentée ci-dessous ?



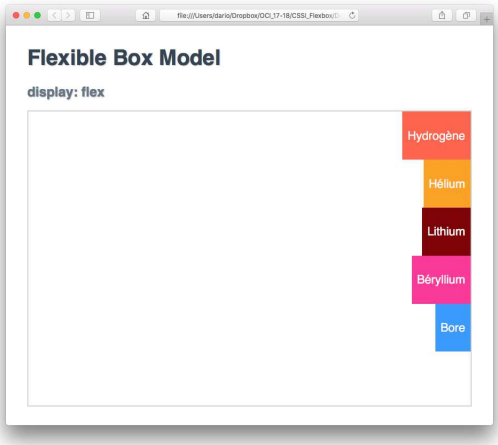
Les divers `item` peuvent passer à la ligne.



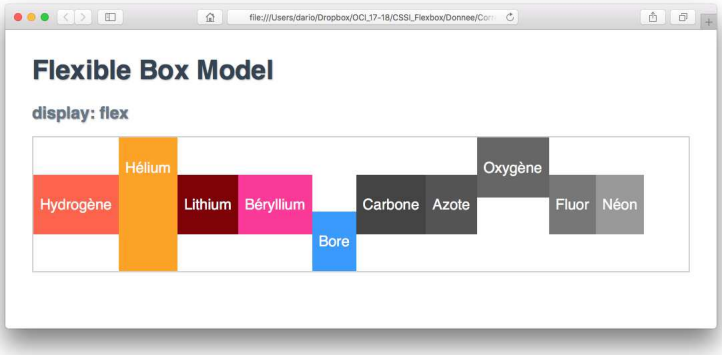
Les alignements dans l'axe de lecture principal sont définis à l'aide de la propriété `justify-content`.



Dans l'axe secondaire, les alignements sont régis via la propriété align-items.



La propriété `align-self`, permet de distinguer l'alignement d'un `flex-item` de ses frères. Les valeurs de cette propriété sont identiques à celles de `align-items`.



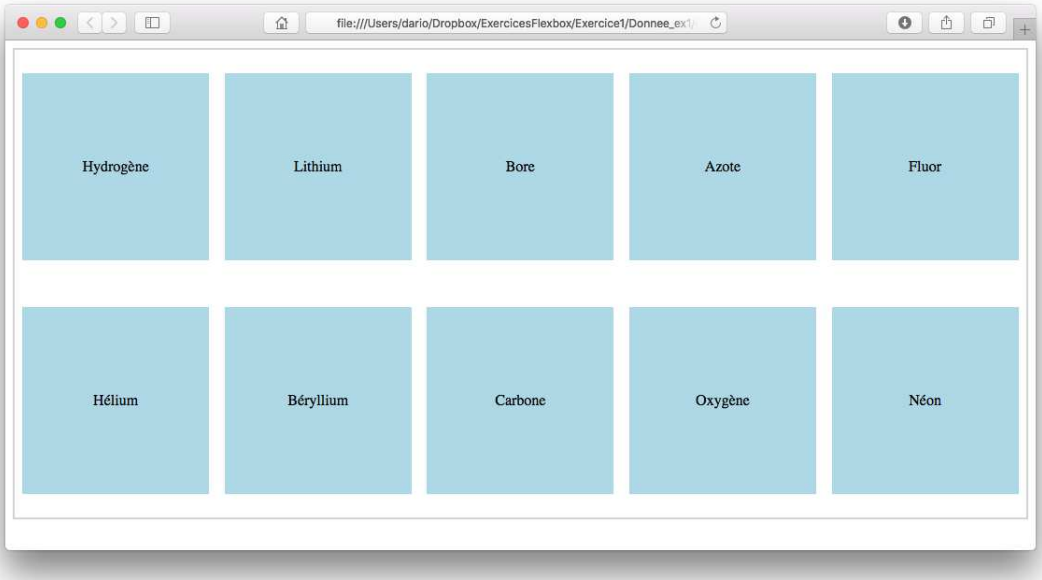
Exercice 5.21

Dans le fichier `ex2_flexbox.html`, créer un lien vers un fichier `style.css`.

Dans ce fichier CSS, voici les instructions à déclarer :

Pour la `<div class="container">` :

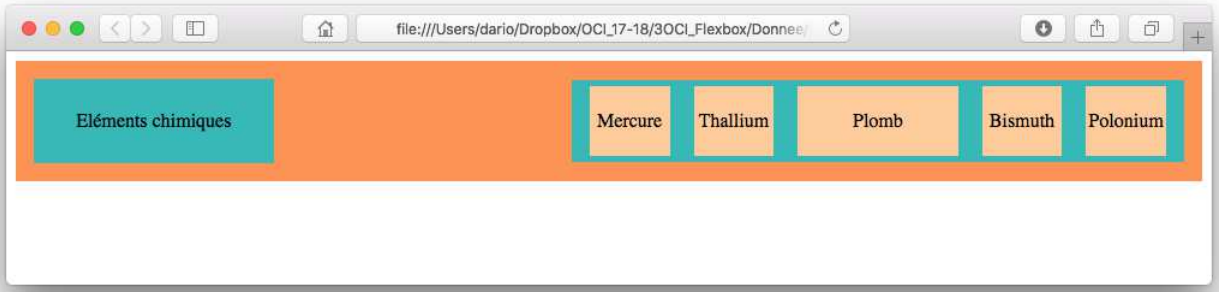
a) Une hauteur de 500px
b) Une bordure de 2px, en trait continu, en gris clair
c) Déclarer la boite comme conteneur "flex"
d) Aligner ses éléments enfants en colonne
e) Ces éléments doivent pouvoir passer à la ligne
f) Ces éléments doivent avoir la même marge de chaque coté (par rapport à l'axe principal et l'axe secondaire)
g) Pour les `<div class="item">` :
   i) Une hauteur et largeur de 200px
   ii) Une couleur de fond "lightblue"



Exercice 5.22

Dans le fichier `ex3_flexbox.html`, créer un lien vers un fichier `style.css`.

Nous voulons obtenir le résultat suivant :



Dans ce fichier CSS, voici les instructions à déclarer :

a) Pour l'élément `<nav>` :
   i) Une hauteur de 100px
   ii) Une couleur de fond `#ff944d`

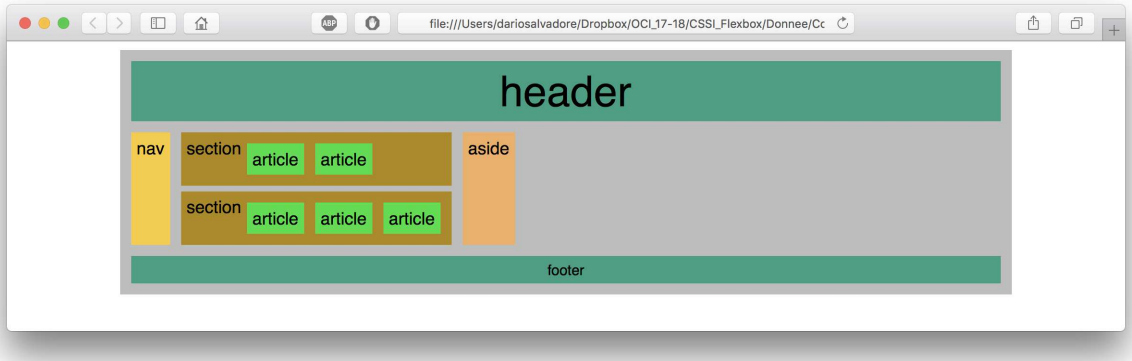
b) Pour l'élément `<div>` :
   i) Une largeur de 200px
   ii) Une marge en haut, à gauche et en bas de 15px
   iii) Une couleur de fond `#2eb8b8`

c) Pour l'élément `<ul>` :
   i) Enlever les puces
   ii) Une marge intérieure de 5px
   iii) Une marge à droite de 15px
   iv) Une couleur de fond `#2eb8b8`

d) Pour l'élément `<li>` :
   i) Une largeur de 80px
   ii) Pas de marge en haut et bas, 10px à droite et à gauche
   iii) Une couleur de fond `#ffcc99`
   iv) L'item *Plomb* est 2 fois plus large que les autres items

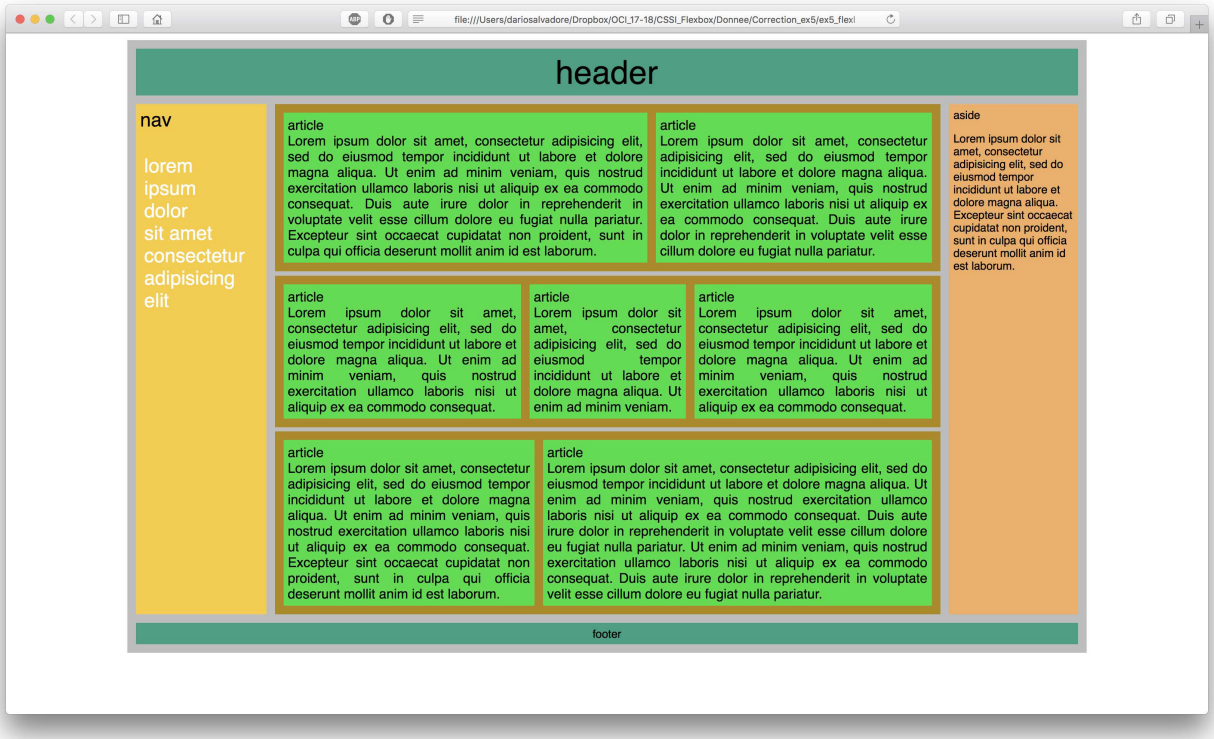
Exercice 5.23

Modifier les fichiers `ex4_flexbox.html` et `style.css` pour obtenir le résultat suivant.



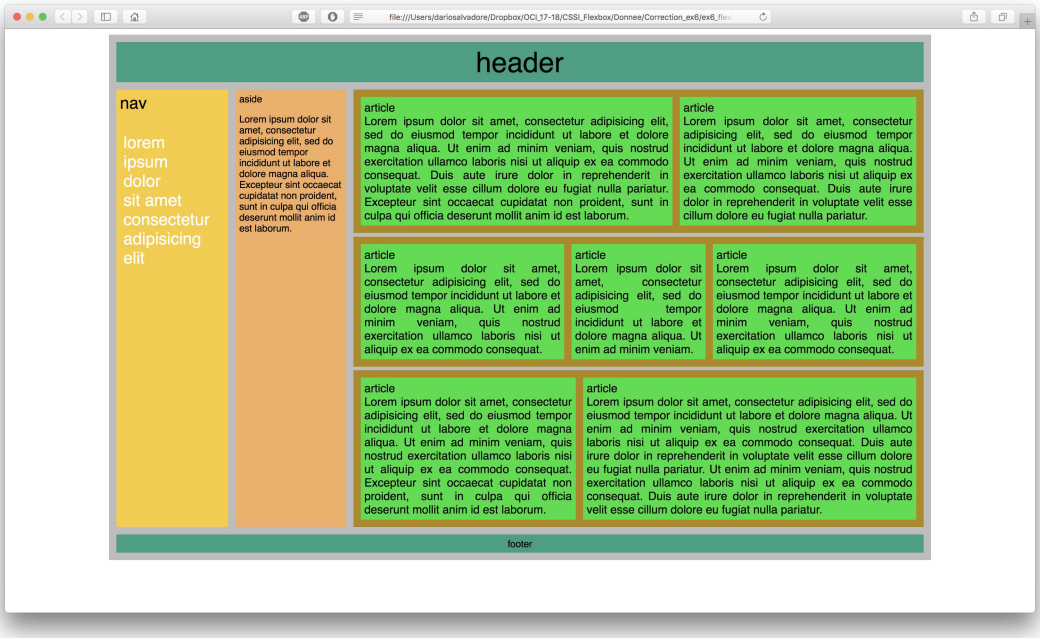
Exercice 5.24

Modifier les fichiers `ex5_flexbox.html` et `style.css` pour obtenir le résultat suivant.



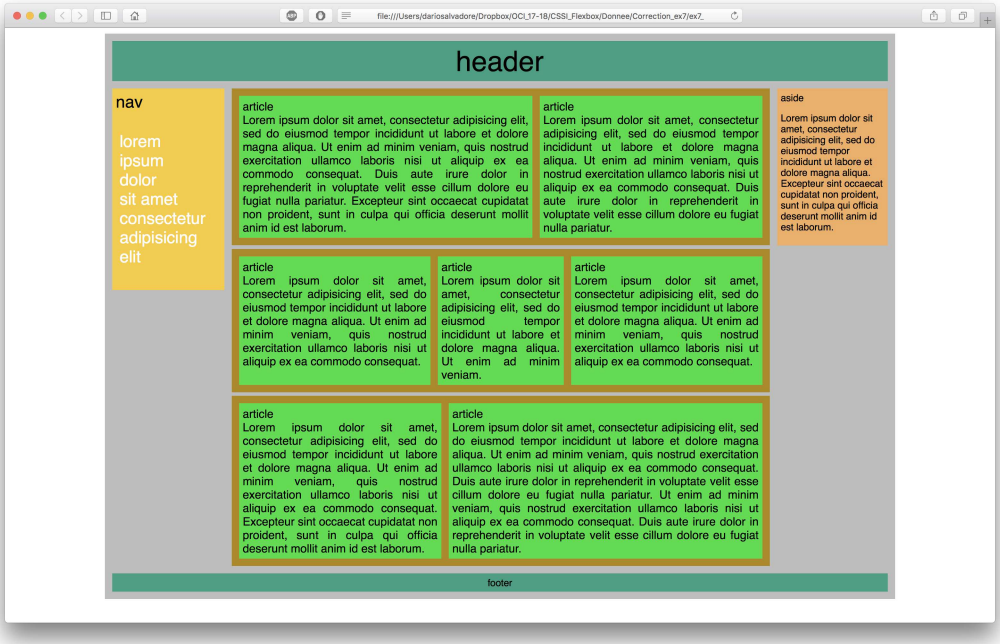
Exercice 5.25

Prendre une copie du fichier css de l'exercice précédent, puis le modifier pour obtenir le résultat ci-dessous.



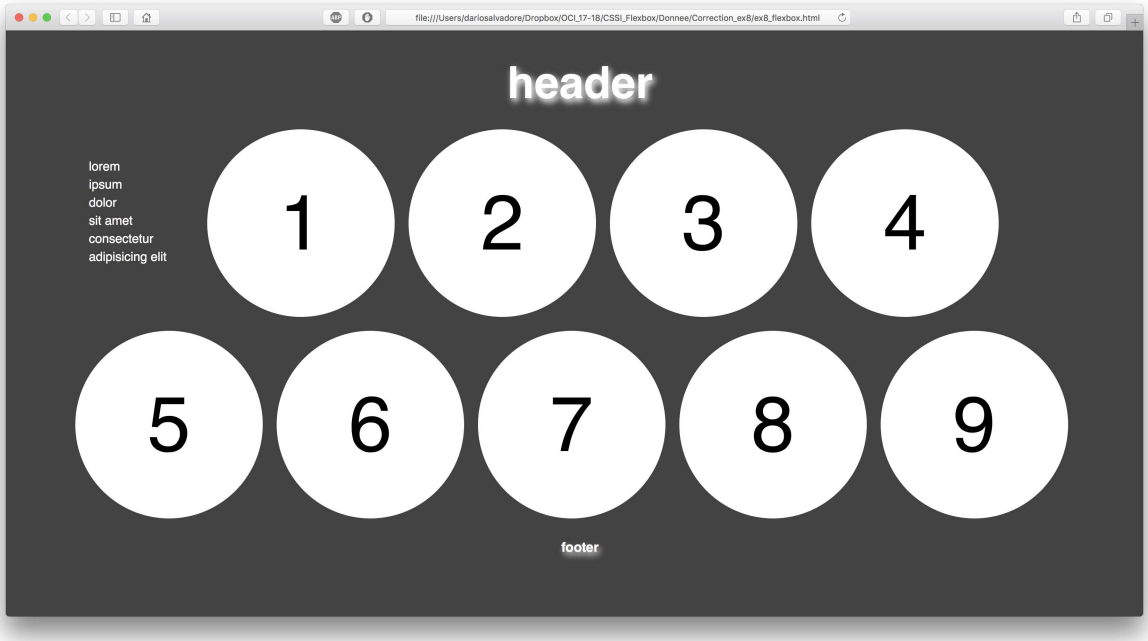
Exercice 5.26

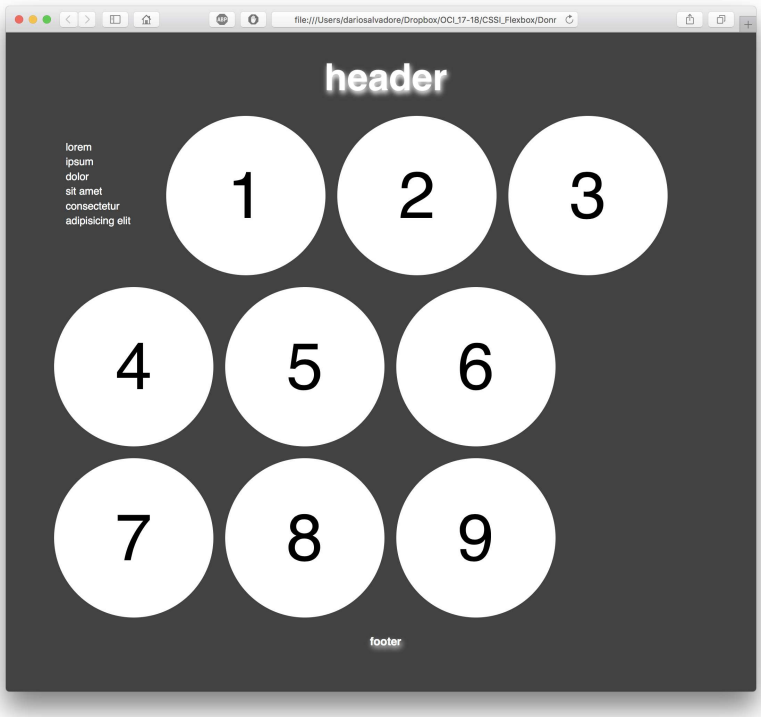
Prendre une copie des fichiers de l'exercice 5, puis les modifier pour obtenir le résultat ci-dessous.



Exercice 5.27

Réaliser la page ci-dessous.





6 Pages PHP

Exercice 6.1

Compléter le fichier `Ex1.php` afin d'obtenir le résultat ci-dessous.

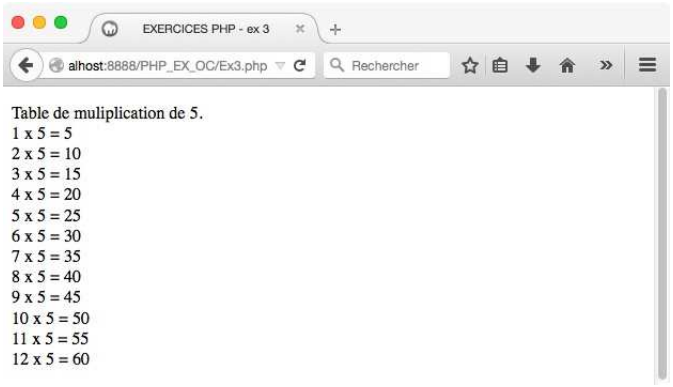
Exercice 6.2

Compléter le fichier `Ex2.php` afin d'obtenir le résultat ci-dessous.

L'affichage dépend de la valeur de la variable `$n`.

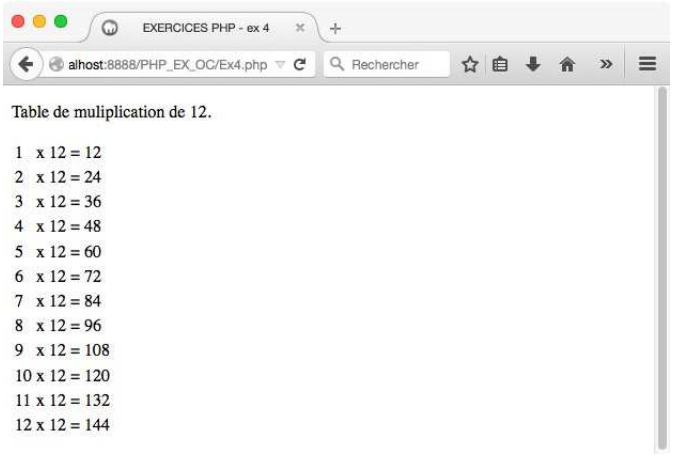
Exercice 6.3

Compléter le fichier `Ex3.php` afin d'obtenir le résultat ci-dessous.



Exercice 6.4

Reprendre l'exercice 3 afin d'obtenir le résultat ci-dessous.



Indication : pour obtenir cet affichage, on peut créer un tableau HTML à la volée.

Exercice 6.5

Choisir un nombre de trois chiffres. Effectuer ensuite des tirages aléatoires et compter le nombre de tirages nécessaire pour obtenir le nombre initial. Arrêter les tirages et afficher le nombre de coups réalisés.

Exercice 6.6

En utilisant la fonction `strlen()`, écrire une boucle qui affiche chaque lettre de la chaîne

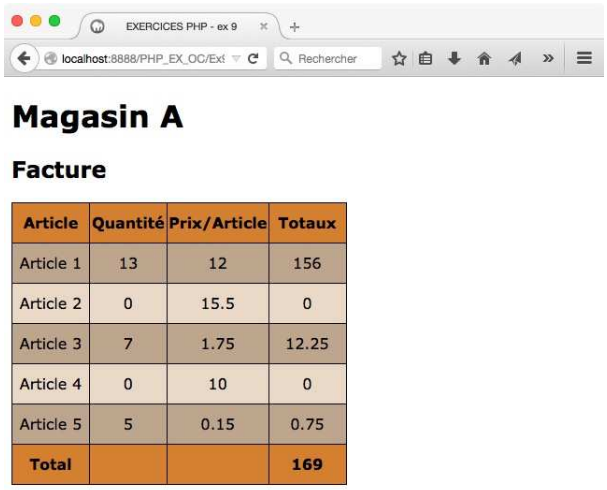
```
Sous le pont Mirabeau coule la Seine
```

sur une ligne différente.

Modifier le programme pour qu'il n'affiche pas les espaces de la chaîne de caractères contenue dans la variable `$texte`.

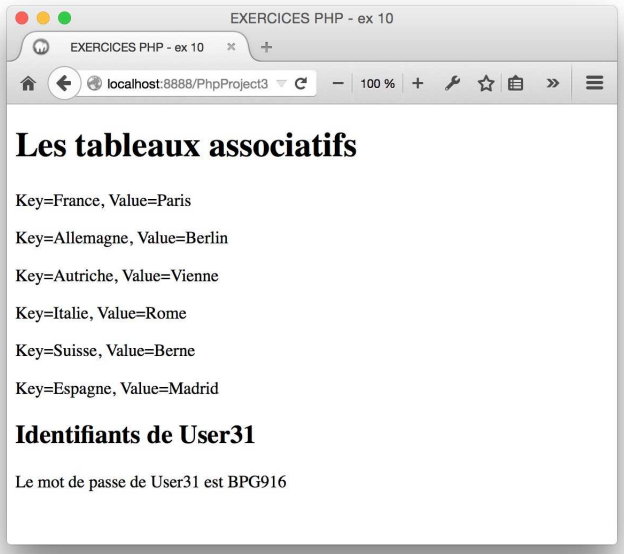
Exercice 6.7

Compléter le fichier `Ex7.php` afin d'obtenir le résultat ci-dessous (mise en page facultative !).



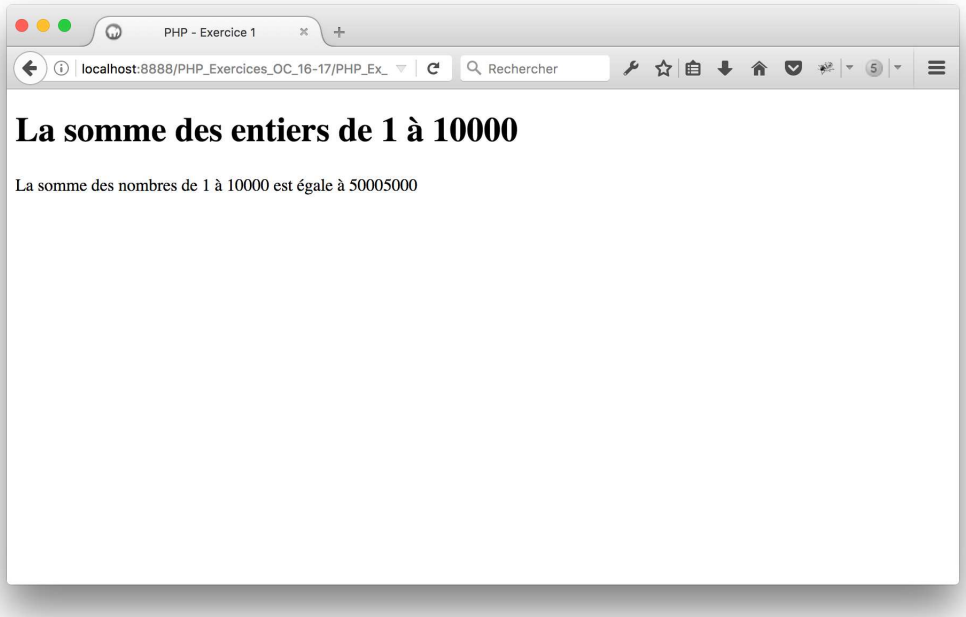
Exercice 6.8

Compléter le fichier `Ex8.php` afin d'obtenir un résultat identique à celui ci-dessous.



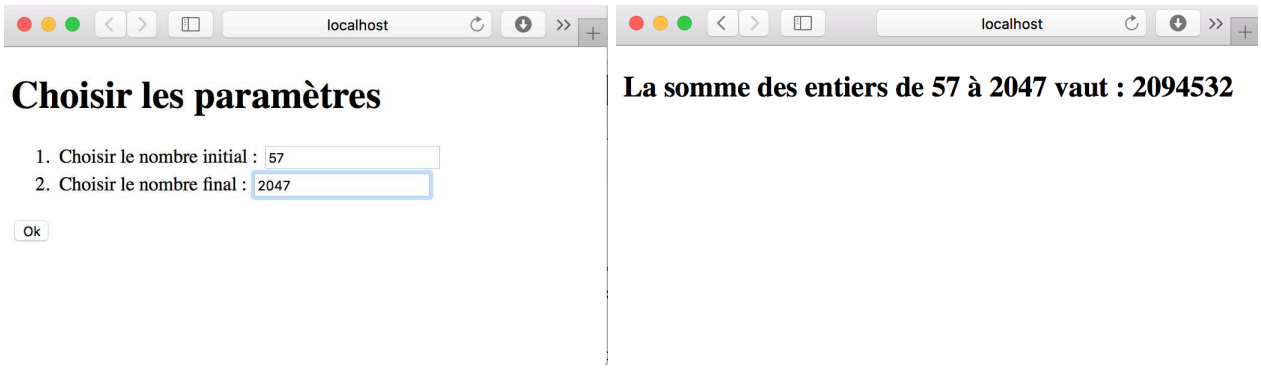
Exercice 6.9

Créer un fichier `Ex9.php` afin d'obtenir le résultat ci-dessous.



Exercice 6.10

Créer un fichier `Ex10.php` et un fichier `Ex10_Traitement.php` afin d'obtenir les deux pages ci-dessous.

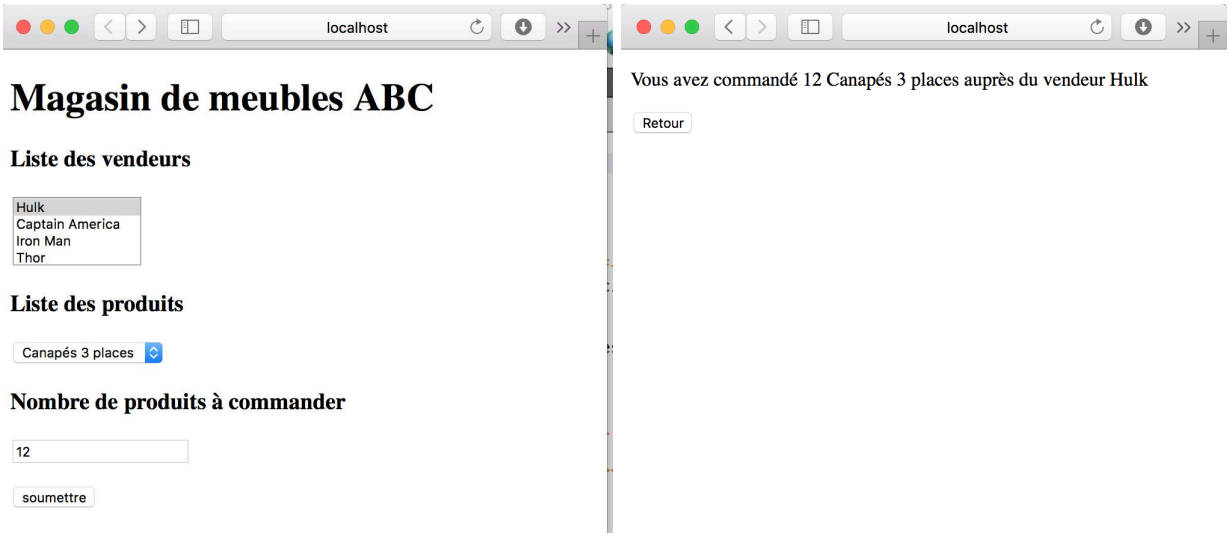


Exercice 6.11

Construire deux pages telles que :

1) La première (`Ex11.php`) affiche :
   a) une liste avec les noms des vendeurs (liste non modifiable)
   b) une liste qui affiche la liste déroulante des produits disponibles
   c) une zone de texte pour saisir le nombre de produits à commander

   La liste des vendeurs et des produits est contenue dans le fichier `produits_vendeurs.php`.
2) La deuxième (`Ex11_Traitement.php`) affiche les données sélectionnées dans la première.

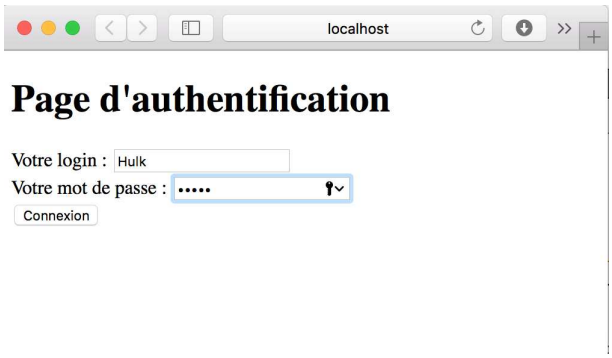


Exercice 6.12

Reprendre l'exercice 11 et indiquer si la saisie n'est pas complète sur la 2ème page.

Exercice 6.13

A partir du fichier `membres.php`, créer la page suivante.



Exercice 6.14

A partir du fichier `membres.php`, créer une page d'authentification comme ci-dessous :