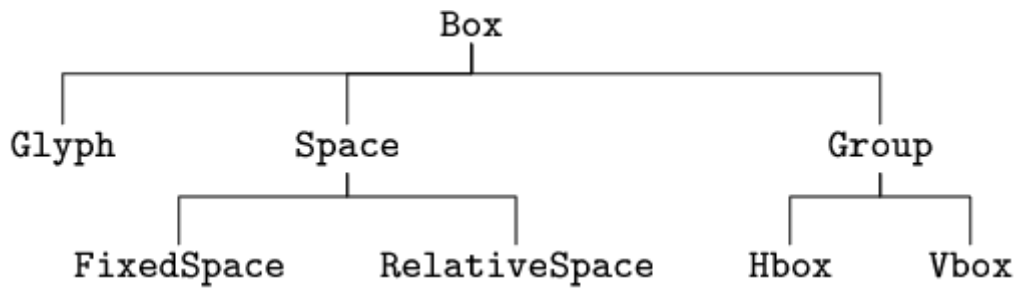
This assignment will be closed on May 23, 2023 (23:59:59).

On May 21, 2023 (23:55:00), we had submissions for the following questions: 1, 2, 3, 4, 5, 6, 7, 8 — Details (/agns/INF371/TD05/2022/uploads/:my/)

Héritage et typographie

Jean-Christophe Filliâtre

Le but de ce TD est de se familiariser avec la notion d'héritage. Nous allons construire progressivement des classes organisées selon l'arbre d'héritage illustré ci-dessous :



Heritage

Ces classes représentent des éléments de typographie.

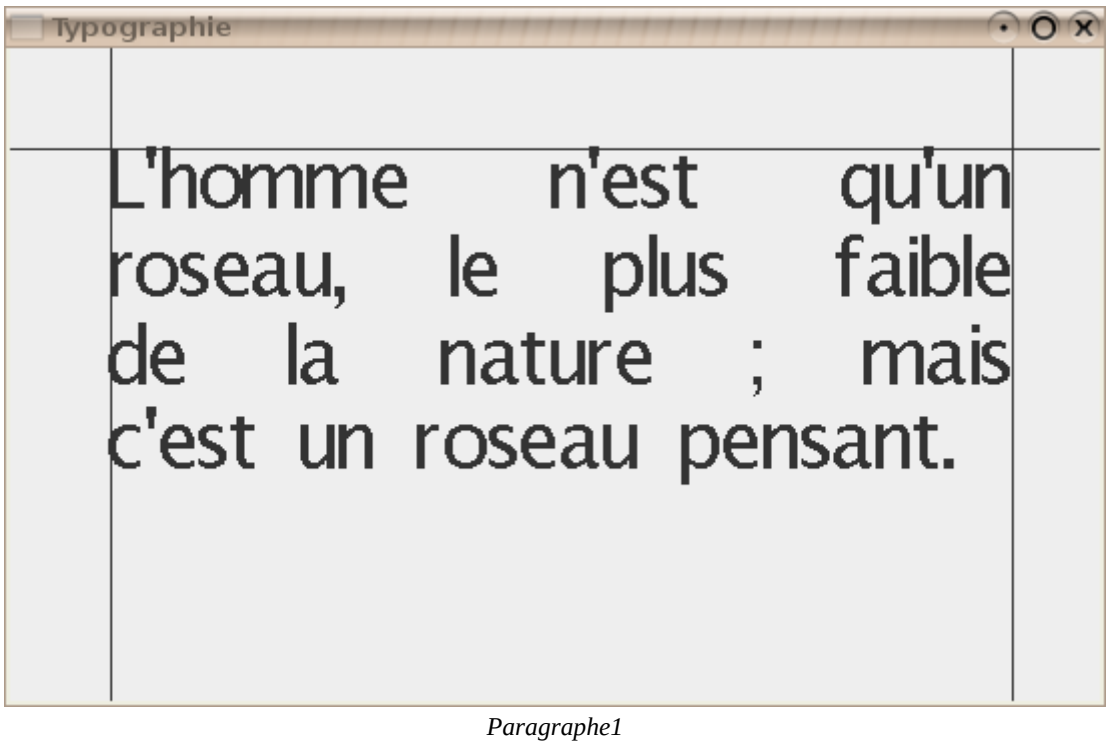
Note pour les utilisateurs d'Eclipse. Dans ce TD nous allons utiliser les bibliothèques Java AWT afin de composer des élements graphiques. Pour faciliter l'accès à ces bibliothèques dans Eclipse, il vaut mieux créer un projet sans fichier module-info.java. Si vous avez un fichier module-info.java et que vous ne souhaitez pas le supprimer, il sera probablement nécessaire de lui ajouter `requires java.desktop;` .

Tests automatiques. Pour les fonctions qui font un affichage graphique, le serveur ne vérifie pas si le dessin est bon. Cependant, au même endroit où vous trouvez habituellement les messages d'erreur, vous aurez accès à un lien `artifacts` qui vous permettra de visualiser côte à côte votre rendu avec un rendu de référence.

Typographie

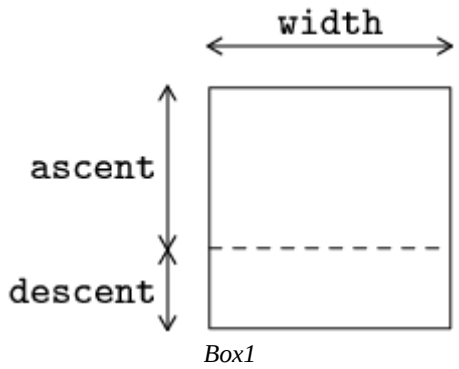
Un document typographié est obtenu en assemblant divers éléments s'apparentant à des boîtes (il seront représentés dans ce TD par des instances de la classe `Box`). Certaines de ces boîtes représentent simplement un unique signe typographique, comme un caractère ; on les appelle des *glyphes* (classe `Glyph`). D'autres boîtes représentent des espaces (classe `Space`) qui peuvent être fixes (espaces horizontaux entre les lettres d'un même mot ; classe `FixedSpace`) ou étirables (espaces entre mots si l'on veut un alignement des lignes qui soit justifié ; classe `RelativeSpace`). Enfin, d'autres boîtes représentent des empilements (classe `Group`) horizontaux (classe `Hbox`) ou verticaux (classe `Vbox`) de boîtes de façon à pouvoir construire des lignes ou des paragraphes.

À titre d'exemple, le resultat suivant (auquel vous devriez arriver en fin de TD) est un empilement vertical de 4 empilements horizontaux. Le premier empilement horizontal est composé de 17 glypes (L, ‘, h, o, m, m, e, n,’, e, s, t, q, u, ’, u, n), 2 espaces étirables et 14 espaces fixes.



Paragraphe1

Une boîte est caractérisée par trois grandeurs : sa largeur, sa hauteur au dessus de la ligne de base (la ligne sur laquelle on écrit le texte) et sa profondeur en dessous de cette même ligne de base. En notant `width`, `ascent` et `descent` ces trois grandeurs, on peut schématiser ainsi une boîte typographique (la ligne de base étant représentée en pointillés) :



Box1

En outre, une quatrième grandeur, `stretchingCapacity`, indique la possibilité, plus ou moins grande, d'étirer la largeur d'une boîte. Si une boîte n'est pas étirable (par exemple, pour une espace fixe), cette quantité est nulle.

Les longueurs `width`, `ascent` et `descent` sont exprimées en points et `stretchingCapacity` est un coefficient. Toutes ces grandeurs seront des flottants (type `double`).

Paquet

La hiérarchie de classes que vous allez définir sera réunie au sein d'un même paquet nommé `typo`. Il faut donc commencer par créer le répertoire éponyme qui contiendra les fichiers de ces classes.

Héritage

Boîte

Comme point de départ, créer une classe abstraite `Box` sur le modèle suivant. Les deux paquets importés seront nécessaires par la suite (question 3).

```java
package typo;

import java.awt.Color;
import java.awt.Graphics;

public abstract class Box {
  // vide pour l'instant
}
```

Cette classe abstraite permettra de déclarer les opérations communes à tous les types de boîte (même si la façon de définir concrètement les opérations dans les sous-classes variera selon les cas).

Question 1

Ajouter des méthodes publiques abstraites `getWidth`, `getAscent`, `getDescent` et `getStretchingCapacity` dans la classe `Box`.

Last submission: May 16, 2023 (16:12:31)

Choose | Choose one or more files... | Submit

Reuse files from previous submissions

Expected files: Box.java

Glyphe

Le premier type d'éléments typographiques que nous allons coder sont les glyphes. Créer une classe `Glyph` héritant de `Box` contenant le code suivant pour les représenter.

```java
package typo;

import java.awt.Font;
import java.awt.Graphics;
import java.awt.font.FontRenderContext;
import java.awt.font.TextLayout;
import java.awt.geom.Rectangle2D;


public class Glyph extends Box {
  final private static FontRenderContext frc
     = new FontRenderContext(null, false, false);
  final private Font font;
  final private char[] chars;
  final private Rectangle2D bounds;

  // classe à compléter (question 2)

  public String toString() {
    return String.format("Glyph(%s)[w=%g, a=%g, d=%g, sC=%g]",
        chars[0], this.getWidth(), this.getAscent(),
        this.getDescent(), this.getStretchingCapacity());
  }
}
```

Ne vous souciez pas de la déclaration de l'attribut `frc` pour le moment. La méthode `toString()` d'un objet `obj` qui n'est pas de type `String` est appelée par défault par `System.out.print` (ou ses variantes). Ainsi, si `g` est un glyphe, on pourra afficher la chaîne de caractères `g.toString()` dans la console à l'aide de la commande `System.out.println(g)`. Cet affichage textuel est utile pour contrôler facilement ce qui est fait.

Question 2

Compléter la classe `Glyph` de la façon suivante :

1. Écrire un constructeur prenant en argument une police de caractères `font` (classe `java.awt.Font` (http://docs.oracle.com/javase/8/docs/api/java/awt/Font.html)) et un caractère `c` (type `char`). On stockera le caractère `c` dans le tableau `chars`, qui sera de taille 1 (ceci afin de faciliter l'utilisation de `drawChars` dans la question suivante).

   Les dimensions `bounds` du glyphe (qui sont déterminées par le choix de la police `font` et du caractère `c`) peuvent être obtenues à l'aide du code suivant :

   ```
   TextLayout layout = new TextLayout("" + chars[0], font, frc);
   bounds = layout.getBounds();
   ```

2. Définissez l'implementation de la méthode `getStretchingCapacity` renvoyant 0 (par définition, la capacité d'étirement d'un glyphe est nulle).

3. Définissez l'implementation des méthodes `getWidth`, `getAscent` et `getDescent` renvoyant les dimensions du glyphe. Elles sont obtenues ainsi :

   ```
   //   ascent  = - bounds.getY()
   //   descent = bounds.getHeight() + bounds.getY()
   //   width   = bounds.getWidth();
   ```

Pour tester, on pourra créer une classe `Test` dans le package par défaut (le parent de `typo`) et y copier le code suivant (à appeler depuis `main`) :

```
static void test2() {
  Font f = new Font("SansSerif", Font.PLAIN, 70);
  Glyph g = new Glyph(f, 'g');
  System.out.println(g);
}
```

qui doit donner la sortie suivante (les valeurs peuvent légèrement différer) :

```
Glyph(g)[w=32.0, a=37.125, d=14.734375, sC=0.0]
```

Last submission: May 16, 2023 (16:13:11)

Choose | Choose one or more files... | Submit

Reuse files from previous submissions

Expected files: Box.java, Glyph.java

Dessin

Nous allons maintenant afficher les résultats à l'aide d'une fenêtre graphique. Pour ce faire, on fournit une classe Page (resources/java/Page.java) à ajouter au package par défaut, à côté de `Test`. Vous n'avez pas besoin de lire son code. Pour que cette classe soit utilisable, plusieurs modifications doivent être apportées aux classes `Box` et `Glyph`.

Question 3

1. Ajouter le code suivant dans la classe `Box` :

```
final static boolean DEBUG = false;
public final boolean draw(Graphics graph, double x, double y, double w) {
  if (DEBUG) {
    graph.setColor(Color.red);
    graph.drawRect((int) x, (int) y, (int) w, (int) (getAscent() + getDescent()));
    graph.setColor(Color.black);
  }
  return doDraw(graph, x, y, w);
}
```

Remarquez la présence de la variable `DEBUG`. La mettre temporairement à `true` pourra être très utile pour déboguer votre code dans les questions suivantes.

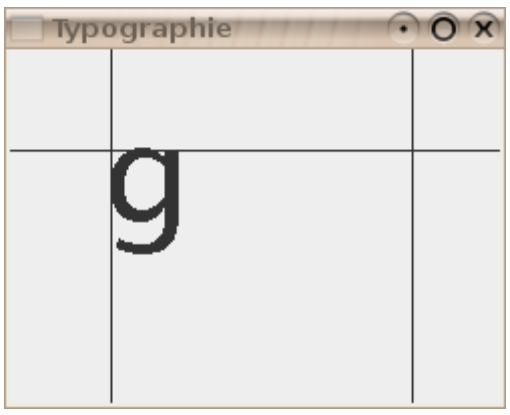
2. Ajouter une méthode `public abstract boolean doDraw(Graphics graph, double x, double y, double w)` dans la classe `Box`. Quelque soit le type de boîte, cette méthode dessinera cette boîte avec son coin supérieur gauche aux coordonnées `x,y` dans un rectangle de largeur `w`. On remarque que les coordonnées Y croissent vers le bas. Le booléen retourné est vrai si et seulement si aucun problème n'a été détecté (il ne pourra être faux que dans le cas des boîtes horizontales, à la question 7). Sa définition concrète sera effectuée dans les sous-classes en fonction du type de boîte à réaliser.

3. Implémentez la méthode `doDraw` dans la classe `Glyph`. Elle doit dessiner le caractère du glyphe à l'emplacement `x,y`. Pour cela, vous pourrez utiliser la méthode `graph.drawChars` (http://docs.oracle.com/javase/8/docs/api/java/awt/Graphics.html#drawChars-char:A-int-int-int-int-), après avoir sélectionné la police avec la méthode `graph.setFont` (http://docs.oracle.com/javase/8/docs/api/java/awt/Graphics.html#setFont-java.awt.Font-). Les coordonnées à passer à `graph.drawChars` (http://docs.oracle.com/javase/8/docs/api/java/awt/Graphics.html#drawChars-char:A-int-int-int-int-) sont `x-bounds.getX(), y-bounds.getY()`. En effet, on veut aligner le coin supérieur gauche du glyphe sur les coordonnées `x,y` mais `drawChars` attend les coordonnées où placer le point d'origine du glyphe. Comme tout est bien fait, `getX` et `getY` donnent les coordonnées du coin supérieur gauche du glyphe relativement à son point d'origine.

On pourra tester avec le code suivant :

```
static void test3() {
    Font f = new Font("SansSerif", Font.PLAIN, 70);
    Glyph g = new Glyph(f, 'g');
    System.out.println(g);
    new Page(g, 150, 150);
}
```

qui doit donner la même sortie que précédemment et l'image suivante :



Test

Last submission: May 16, 2023 (16:20:30)

Choose | Choose one or more files... | Submit

Reuse files from previous submissions

Expected files: Box.java, Glyph.java

Factorisation

Pour contrôler ce que nous faisons, nous avons choisi d'afficher dans la console les caractéristiques des glyphes sous la forme `Glyph(g)[w=32.0, a=37.125, d=14.734375, sC=0.0]`. Nous allons maintenant considerer d'autres sous-classes de `Box` et nous afficherons toujours dans la console leurs caractéristiques sous une forme similaire. Pour gagner en temps et en praticité, nous allons *factoriser* le code commun aux fonctions `toString` de toutes les sous-classes de `Box`.

Question 4

1. Ajouter une méthode `toString` à la classe `Box`, qui renvoie la partie entre crochets (inclus) de la chaîne, c'est-à-dire `[w=32.0, a=37.125, d=14.734375, sC=0.0]` dans l'exemple précédent.

2. Simplifier ensuite le code de la méthode `toString` de `Glyph` en utilisant `super.toString`.

On pourra réutiliser `test3` pour tester.

Last submission: May 16, 2023 (16:26:58)

Choose | Choose one or more files... | Submit

Reuse files from previous submissions

Expected files: Box.java, Glyph.java

Espace

Les espaces peuvent être soit des espaces fixes (entre deux caractères d'un même mot), soit des espaces étirables (entre les mots d'une ligne justifiée).

Question 5

1. Définir une classe `Space` héritant de `Box` pour représenter une espace ayant comme attributs une largeur `width` et une capacité d'étirement `stretchingCapacity`, tout deux de type `double`. Quelle doit être la visibilité de ces attributs ?

2. Définir un constructeur pour la classe `Space` prenant en arguments une dimension minimale et une capacité d'étirement. Définir la méthode `doDraw` comme ne dessinant rien (il n'y a rien à dessiner pour représenter une espace) et les méthodes `getAscent` et `getDescent` comme retournant 0. Redéfinir aussi la méthode `toString` pour indiquer qu'il s'agit d'une espace, en renvoyant la chaîne `Space[w=...]` et en utilisant les informations fournies par `super.toString`.

3. Définir ensuite deux sous-classes `FixedSpace` et `RelativeSpace` de `Space`, représentant respectivement une espace fixe et une espace proportionnelle à la taille d'une police de caractères. Le constructeur de `FixedSpace` prendra une dimension de type `double` en argument. Le constructeur de `RelativeSpace` prendra un coefficient `c` de type `double` et une police de caractères `f` de type `Font`, pour construire une espace de dimension `c * f.getSize()` et de capacité d'étirement 1. Ces deux constructeurs feront appel au constructeur de la classe `Space`, avec la syntaxe `super(...);`. On ne demande pas de redéfinir `toString` dans ces deux classes.

4. Écrire une méthode `test5` dans la classe `Test`, qui construit trois objets dans les classes `Space`, `FixedSpace` et `RelativeSpace`, et les affiche avec `System.out.println`. On doit obtenir quelque chose comme suit:

```
Space[w=2.0, a=0.0, d=0.0, sC=3.0]
Space[w=5.0, a=0.0, d=0.0, sC=0.0]
Space[w=35.0, a=0.0, d=0.0, sC=1.0]
```

Last submission: May 16, 2023 (16:43:40)

Choose | Choose one or more files... | Submit

Reuse files from previous submissions

Expected files: Box.java, FixedSpace.java, RelativeSpace.java, Space.java

Groupe

Les groupes sont des boîtes “conteneurs” permettant d’empiler horizontalement (`Hbox`) ou verticalement (`Vbox`) des boîtes.

Note : par définition, un groupe est une structure inductive.

Question 6

1. Définir une classe abstraite `Group` héritant de `Box` pour représenter un empilement de boîtes. Cette classe va être utilisée pour factoriser autant que possible le code commun aux deux types d’empilements (horizontaux ou verticaux). On pourra utiliser une `LinkedList` (http://docs.oracle.com/javase/8/docs/api/java/util/LinkedList.html) (du package `java.util`) pour représenter un empilement comme suit :

```
protected final LinkedList<Box> list = new LinkedList<Box>();
```

Note : `LinkedList<E>` est une classe générique qui implemente des listes chaînées d’éléments de type `E` . L’élément `<E>` est un *placeholder* pour la classe réelle qui est utilisée. Différentes classes peuvent être utilisées pour créer différents types de noeuds. Par exemple, dans TD3:4 vous avez créé la classe `WordList` qui implémente une liste chaînée de `String` et la classe `EntryList` qui implémente une liste chaînée de `Entry` . Au lieu de ces classes vous auriez pu utiliser `LinkedList<String>` et `LinkedList<Entry>` .

2. Étendre la classe `Group` avec une méthode `add` qui permet d’ajouter une nouvelle boîte à la fin de leur liste.

3. Définir les méthodes `getWidth` , `getAscent` , `getDescent` et `getStretchingCapacity` de la classe `Group` de telle sorte qu’elles renvoient les valeurs mémorisées dans les quatre champs `ascent` , `descent` , `width` et `stretchingCapacity` .

Note : à ce point, on ne peut pas encore calculer les valeurs de ces champs car elles dépendent du fait que les espaces soient fixes ou étirables.

4. Redéfinir la méthode `toString` de la classe `Group` pour afficher un groupe sous la forme suivante.

```
[w=...]{
   boite 1,
   ...
   boite n,
}
```

On rappelle qu’on peut parcourir les éléments de la liste `list` avec la syntaxe `for (Box b: list) ...` . On rappelle également qu’on peut insérer un retour-chariot dans une chaîne de caractères avec la syntaxe `\n` . On pourra également utiliser la méthode `replaceAll(str1,str2)` de la classe `String` permettant de remplacer chaque occurence de `str1` par `str2` dans une chaîne de caractères.

On veillera à ce que les indentations de boite 1, ..., boite n soient conservées, de sorte à obtenir par exemple :

```
[w=...]{
   [w=...]{
      boite 1.1,
      ...
      boite 1.n,
   },
   ...
   boite n,
}
```

5. Écrire une classe `TestableGroup` héritant de la classe groupe et définissant une méthode `doDraw` ne dessinant rien. Ajouter ensuite une méthode `test6` dans la classe `Test` qui produit quelque chose comme

```
[w=0.0, a=0.0, d=0.0, sC=0.0]{
   [w=0.0, a=0.0, d=0.0, sC=0.0]{
      Space[w=2.0, a=0.0, d=0.0, sC=3.0],
      Space[w=5.0, a=0.0, d=0.0, sC=0.0],
   },
   Space[w=35.0, a=0.0, d=0.0, sC=1.0],
}
```

Last submission: May 16, 2023 (17:11:48)

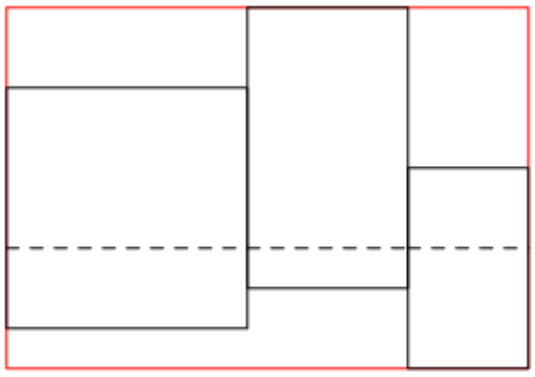
Choose | Choose one or more files... | Submit

Reuse files from previous submissions

Expected files: Box.java, FixedSpace.java, Glyph.java, Group.java, RelativeSpace.java, Space.java, TestableGroup.java

Boîte horizontale

Une boîte horizontale est un groupe représentant un empilement horizontal de boîtes (ordonnées de la gauche vers la droite). Toutes les composantes partagent la même ligne de base. Ainsi, une boîte horizontale contenant trois boîtes ressemble à ceci (où la boîte horizontale est matérialisée en rouge) :



Hbox

On définit la capacité d'étirement de la boîte comme la somme des capacités d'étirement de ses composantes.

Note : Pour vous familiariser avec l'héritage, nous vous demandons de ne pas utiliser `instanceof` dans ce TD. Les propriétés de la redéfinition de fonction sont parfaitement adaptées à ce sujet.

Question 7

1. Définir la classe `Hbox` héritant de `Group` et représentant les boîtes horizontales.

2. Redéfinir la méthode `add` pour qu'elle remplisse la liste avec `super.add` et mette à jour les quatre champs `ascent`, `descent`, `width` et `stretchingCapacity`.

3. Redéfinir la méthode `toString` pour indiquer qu'il s'agit d'une boîte horizontale.

4. Écrire la méthode `doDraw` permettant de dessiner une boîte horizontale. Elle procède de la manière suivante. Le dernier paramètre de `doDraw`, `w`, spécifie la largeur désirée. La largeur minimale est obtenue par `getWidth` ; appelons-la `mw`. Si `mw > w` alors la boîte ne peut pas tenir dans la largeur `w`. On la dessine alors tout de même (sur une largeur `mw`), mais `doDraw` retourne alors le booléen `false`. Si en revanche `w >= mw`, alors il n'y a pas de problème et on va répartir la différence `w-mw` sur toutes les espaces étirables contenues à l'intérieur de la boîte horizontale, proportionnellement à la capacité d'étirement de chacun. Si par exemple la boîte contient deux espaces de capacités 1 et 2 respectivement, alors un tiers de l'espace supplémentaire sera donné au premier et deux tiers au second. Comme on a justement attribué aux glyphes une capacité d'étirement 0, le traitement peut être fait de manière uniforme, sans avoir à connaître la nature de chaque composante.

5. On pourra tester cette classe avec le code suivant

```
static void test7a() {
  Hbox h = new Hbox();
  System.out.println(h);
  Font f = new Font("SansSerif", Font.PLAIN, 40);
  h.add(new Glyph(f, 'a'));
  System.out.println(h);
  h.add(new Space(2., 3.));
  System.out.println(h);
}
```

qui doit donner une sortie de la forme

```
Hbox[w=0.0, a=0.0, d=0.0, sC=0.0]{
}
Hbox[w=19.09375, a=21.21875, d=0.46875, sC=0.0]{
   Glyph(a)[w=19.09375, a=21.21875, d=0.46875, sC=0.0],
}
Hbox[w=21.09375, a=21.21875, d=0.46875, sC=3.0]{
   Glyph(a)[w=19.09375, a=21.21875, d=0.46875, sC=0.0],
   Space[w=2.0, a=0.0, d=0.0, sC=3.0],
}
```

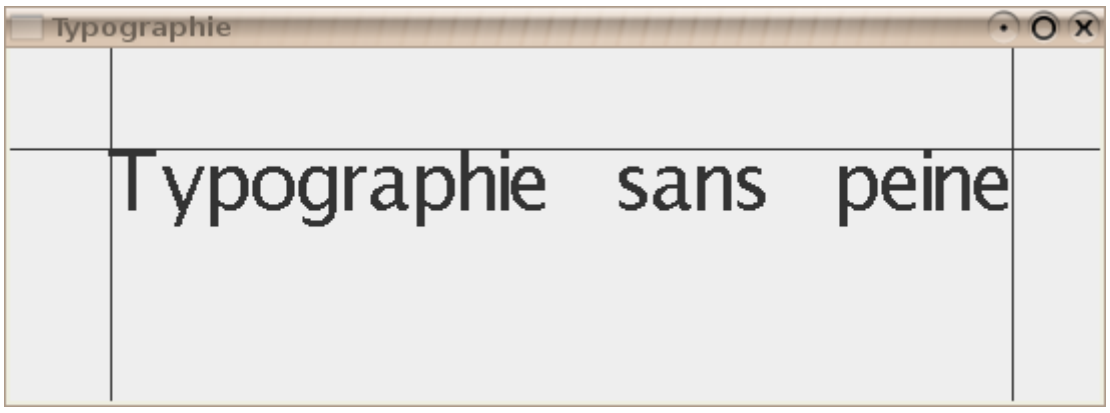
On pourra ensuite la tester avec le code suivant

```
static Hbox lineFromString(Font f, String s) {
  Hbox line = new Hbox();
  for (int i = 0; i < s.length(); i++) {
    char c = s.charAt(i);
    if (c == ' ')
      line.add(new RelativeSpace(0.5, f));
    else {
      line.add(new Glyph(f, c));
      if (i < s.length() - 1 && s.charAt(i+1)!=' ')
        line.add(new FixedSpace(2));
    }
  }
  return line;
}

static void test7b() {
  Font f = new Font("SansSerif", Font.PLAIN, 40);
  Box t = lineFromString(f, "Typographie sans peine");
  System.out.println(t);
  new Page(t, 450, 150);
}
```

qui doit donner une sortie de la forme

```
Hbox[w=410.359375, a=28.640625, d=8.421875, sC=2.0]{
   Glyph(T)[w=22.703125, a=28.640625, d=0.0, sC=0.0],
   Space[w=2.0, a=0.0, d=0.0, sC=0.0],
   Glyph(y)[w=19.015625, a=20.75, d=8.421875, sC=0.0],
   ...
```

et le résultat suivant :



Ligne

Enfin, ce dernier test peut révéler des erreurs d’arithmétique :

```
static void test7c() {
  Font f = new Font("SansSerif", Font.PLAIN, 40);
  Box t = lineFromString(f, "Test");
  System.out.println(t);
  new Page(t, 450, 150);
}
```

Last submission: May 21, 2023 (17:24:37)

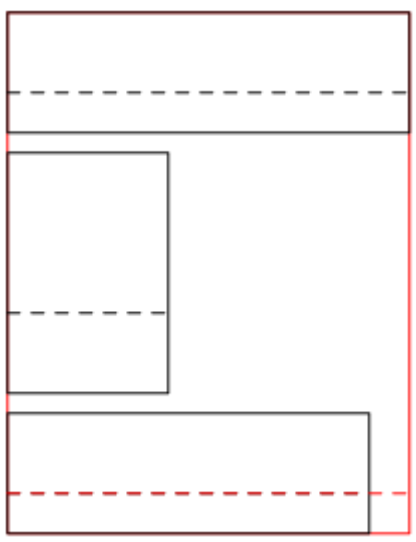
Choose | Choose one or more files... | Submit

Reuse files from previous submissions

Expected files: Box.java, FixedSpace.java, Glyph.java, Group.java, Hbox.java, RelativeSpace.java, Space.java

Boîte verticale

Une boîte verticale est un groupe représentant un empilement vertical de boîtes (ordonnées du haut vers le bas). Les différentes boîtes qu’elle contient sont séparées par un interligne. Par définition, la ligne de base d’une boîte verticale est celle de sa boîte la plus basse. Ainsi, une boîte verticale contenant trois boîtes ressemble à ceci (où la boîte verticale est matérialisée en rouge) :



Vbox

On définit la capacité d’étirement de la boîte verticale comme le maximum des capacités d’étirement de ses composantes.

Question 8

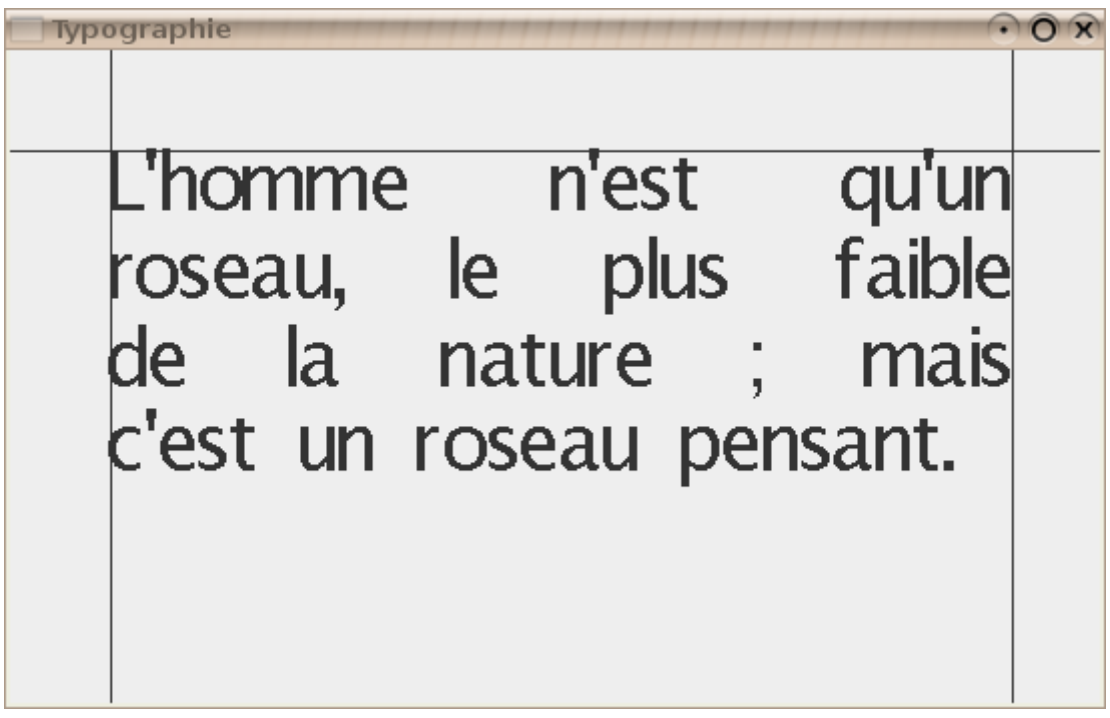
1. Définir une classe `Vbox` héritant de la classe `Group` et représentant un empilement vertical de boîtes. Ajouter un constructeur prenant en argument un interligne `lineSkip` de type `double`.
2. Redéfinir la méthode `add` pour qu’elle remplisse la liste avec `super.add` et mette à jour les quatre champs `ascent`, `descent`, `width` et `stretchingCapacity`.
3. Redéfinir la méthode `toString` pour indiquer qu’il s’agit d’une boîte verticale.

4. Écrire la méthode `doDraw` permettant de dessiner une boîte verticale. Pour cela, il suffit de dessiner les boîtes les unes au dessus des autres, en partant du haut et en espaçant les boîtes de la dimension indiquée par `lineSkip` . On rappelle que les coordonnées Y croissent vers le bas.

5. Tester votre programme avec le code suivant

```
final static Box hfill = new Space(0, Double.POSITIVE_INFINITY);

static Vbox fromString(Font f, String s) {
  Vbox     text  = new Vbox(5);
  String[] lines = s.split("\n");

  for (int i = 0; i < lines.length; ++i) {
    Hbox line = lineFromString(f, lines[i]);

    if (i+1 == lines.length)
      line.add(hfill);
    text.add(line);
  }
  return text;
}

static void test8a() {
  Font f = new Font("SansSerif", Font.PLAIN, 40);
  Box  t = fromString(f,
    "L'homme n'est qu'un\n" +
    "roseau, le plus faible\n" +
    "de la nature ; mais\n" +
    "c'est un roseau pensant.");
  new Page(t, 450);
}
```

qui doit donner le résultat suivant :



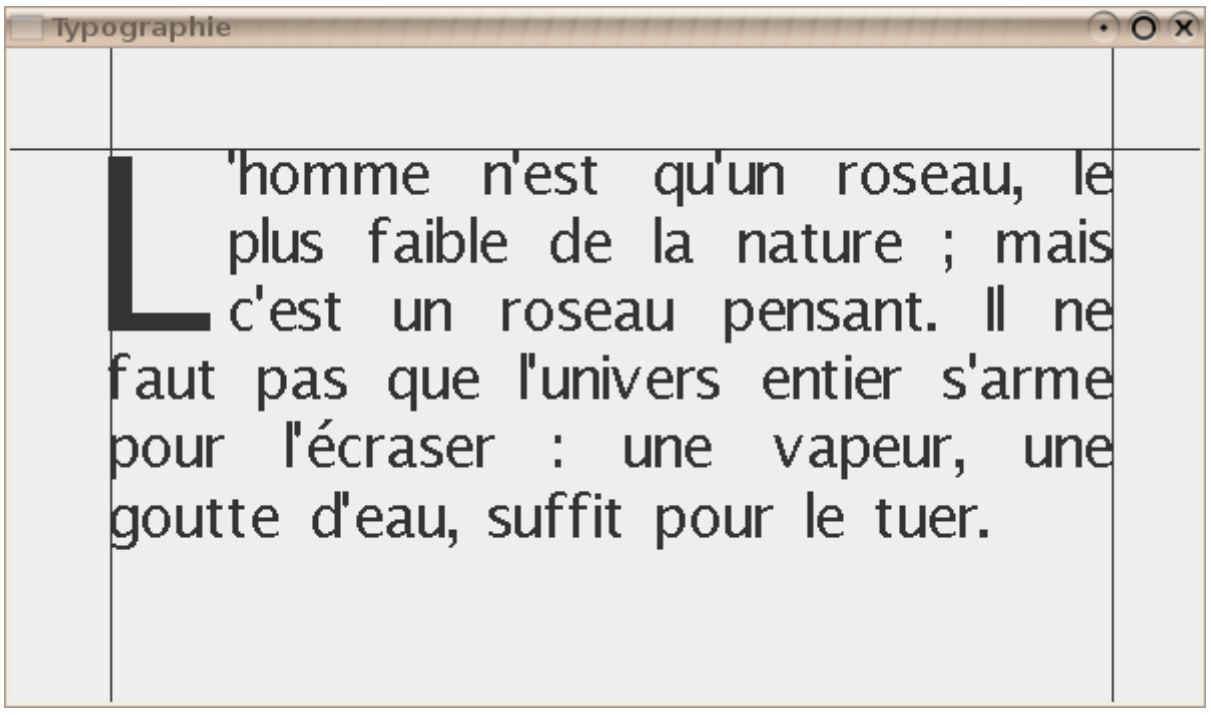
Paragraphe1

---

ou encore avec le code suivant qui dessine une lettrine :

```
static void test8b() {
  Font f = new Font("SansSerif", Font.PLAIN, 30);
  Font lettrinef = new Font("SansSerif", Font.PLAIN, 120);
  Vbox t = new Vbox(5);
  Hbox h = new Hbox();
  h.add(new Glyph(lettrinef, 'L'));
  h.add(new Space(3, 1));
  Vbox l = new Vbox(5);
  l.add(lineFromString(f, "'homme n'est qu'un roseau, le"));
  l.add(lineFromString(f, "plus faible de la nature ; mais"));
  l.add(lineFromString(f, "c'est un roseau pensant. Il ne"));
  h.add(l);
  t.add(h);
  t.add(lineFromString(f, "faut pas que l'univers entier s'arme"));
  t.add(lineFromString(f, "pour l'écraser : une vapeur, une"));
  t.add(fromString(f, "goutte d'eau, suffit pour le tuer."));
  new Page(t, 500);
}
```

qui doit ressembler à ceci :



Lettrine

Last submission: May 21, 2023 (23:53:48)

Choose | Choose one or more files... | Submit

Reuse files from previous submissions

Expected files: Box.java, FixedSpace.java, Glyph.java, Group.java, Hbox.java, RelativeSpace.java, Space.java, Vbox.java