

Programmation

SIE/GCG, Cours I I

6 mai 2024

Jean-Philippe Pellet

```

class Progwin(Canvas):
    def __init__(self, parent, title) -> None:
        Canvas.__init__(self, parent, height=2 * TOP_MARGIN + 4 * LINE_HEIGHT, highlightthickness=0, title=title)

    def redraw(
        self,
        program: Program,
        current_subprogram: List[Instruction],
        current_instruction_index: int,
    ) -> None:
        height = self.winfo_height()
        width = self.winfo_width()

        self.delete(ALL)
        self.create_rectangle(0, 0, width, height, fill=window_background_color, width=0)

        # boucle pour les 4 sous-programmes P1 à P4
        for i, subprogram in enumerate(
            [program.P1, program.P2, program.P3, program.P4]
        ):
            # dessin du titre
            instruction_center_y = TOP_MARGIN + 1 * LINE_HEIGHT + LINE_HEIGHT // 2
            self.create_text(LEFT_MARGIN // 3, instruction_center_y, text=f"P{i + 1}")

            # dessin de chaque instruction
            for j, instr in enumerate(subprogram):
                instruction_center_x = (
                    LEFT_MARGIN
                    + j * (INSTRUCTION_BOX_SPACING + INSTRUCTION_BOX_WIDTH)
                    + INSTRUCTION_BOX_WIDTH // 2
                )
                instruction_center_y = TOP_MARGIN + 1 * LINE_HEIGHT + LINE_HEIGHT // 2
                self.create_text(instruction_center_x, instruction_center_y, text=instr)
            
```

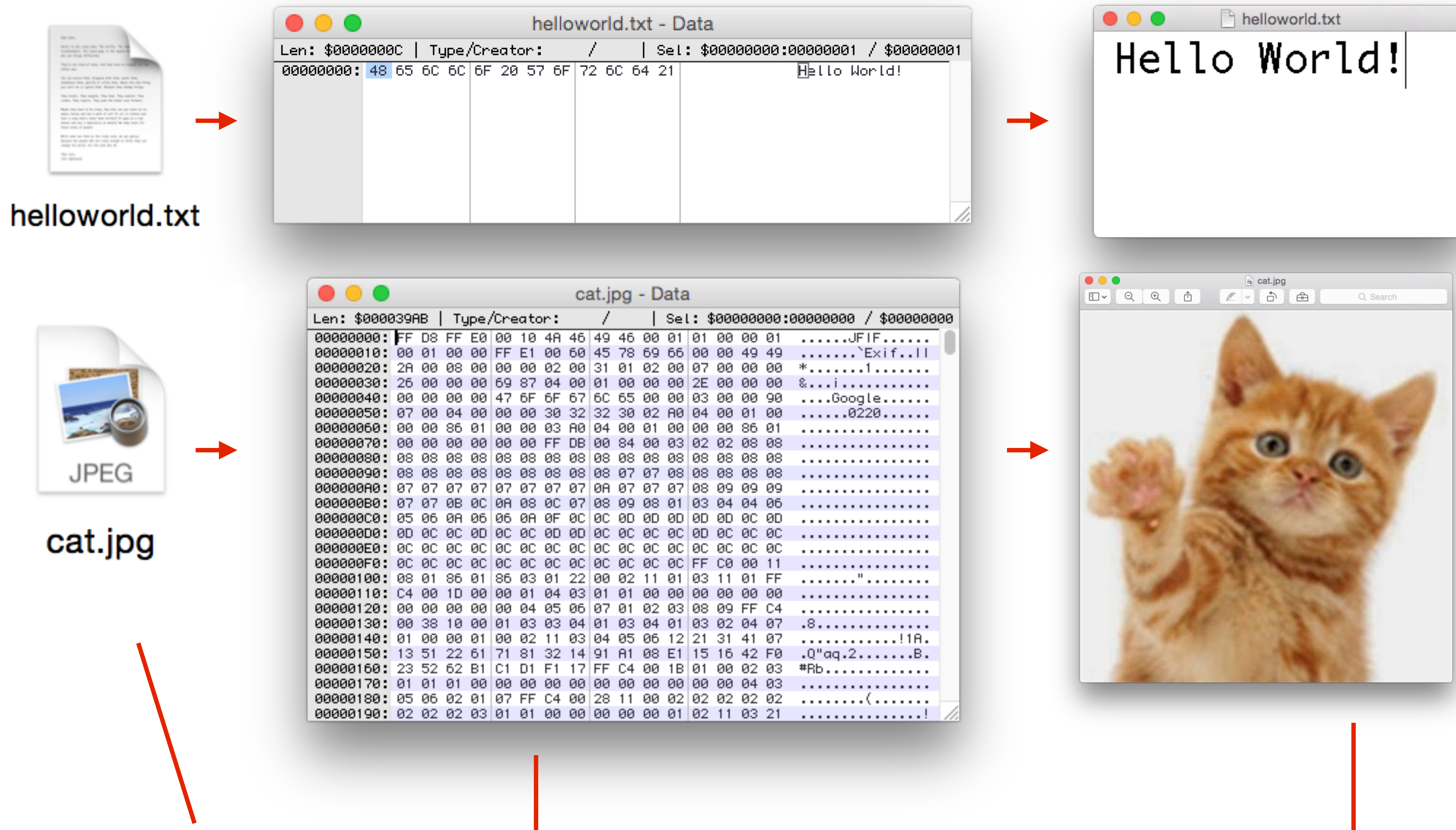
Previously, on ICC Programmation...

- **Types** de base en Python: `int`, `float`, `str`, `bool`
- **Méthodes, fonctions et slicing** pour calculer des valeurs dérivées
- **Conditions** pour exécuter du code selon la valeur d'une expression booléenne:
`if <condition>: ... else: ...` et ses variantes
- **Boucles** pour exécuter du code plusieurs fois:
- **Déclaration de fonctions** avec type de retour et paramètres:
— `def calculate_area(r: float) -> float: return ...`
- Utilisation de **listes, sets** et **dictionnaires**
- Déclaration de **classes**: `@dataclass class Rectangle: ...`
- Création, chargement, manipulation et sauvegarde d'**images**
- **Programmation dynamique** pour trouver des seams
- Fonctions comme **valeurs, paramètres**; fonctions **d'ordre supérieur**

Cours de cette semaine

Lecture/écriture de fichiers texte
Opérations sur des chaînes de caractères

Fichiers et bytes



Un **fichier** est une **séquence de bytes** qui, interprétés correctement, ont **du sens**

Les fichiers texte

une *séquence de bytes* (1 byte = 8 bits)

0	1	2	3	4	5	6	7	8	9	10	11	index du byte
01001000	01100101	01101100	01101100	01101111	00100000	01010111	11011111	01110010	01101100	01100100	00100001	notation binaire
48	65	6C	6C	6F	20	57	6F	72	6C	64	21	notation hexadécimale
H	e	l	l	o		W	o	r	l	d	!	interprétation texte

+ quelques *informations supplémentaires*:

*nom, extension/format, taille, droits d'accès, date de création,
date de la dernière modification, ..., position physique sur le disque*

Le système qui *interprète* chaque byte (ou séquence de bytes) pour retrouver le caractère correspondant repose sur un certain *encodage des caractères*

ASCII Charset

American Standard Code for Information Interchange (1963)

0	Null character	65	“A”
1	Start of header	66	“B”
...		67	“C”
9	Tabulator (“\t”)
10	Line feed (“\n”)	90	“Z”
...		...	
32	Space	97	“a”
...	
48	“0”	122	“z”
...	
57	“9”	127	Delete

Manquent: à, é, ä, €, ...

Unicode, UTF-8 Charset

<http://fr.wikipedia.org/wiki/Unicode>

<http://fr.wikipedia.org/wiki/UTF-8>

Unicode:

Liste avec tous les caractères qui existent,
y compris les caractères chinois, japonais, ...
(env. 1 million théoriquement, env. 120'000 utilisés)

UTF-8:

Format de codage des signes Unicode;
utilise **entre 1 et 4 bytes** pour un caractère

Tableau des caractères Unicode

The screenshot shows the 'Characters' application window. On the left is a sidebar with various character categories. The main area displays a table of characters with columns for Unicode, Title, and Category. Below this is a grid of characters, with the Cyrillic section selected. The grid shows characters from U+0440 to U+04E0, including Cyrillic letters with diacritics and special characters.

Unicode	Title	Category
00000180	Latin Extended-B	Latin
00000250	IPA Extensions	Latin
00000280	Spacing Modifier Letters	Modifier Letters
00000300	Combining Diacritical Marks	Combining Marks
00000370	Greek and Coptic	Greek
00000400	Cyrillic	Cyrillic
00000500	Cyrillic Supplement	Cyrillic
00000530	Armenian	Armenian
00000590	Hebrew	Hebrew
00000600	Arabic	Arabic
00000700	Syriac	Syriac
00000750	Arabic Supplement	Arabic

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0440	Cyrillic	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ъ	Ы	Ь	Э	Ю	Я
0450	è	ë	ђ	ѓ	є	ѕ	і	ї	ј	љ	њ	ћ	ќ	й	ў	џ
0460	Ω	ω	ѣ	ѣ	Є	є	А	а	І	і	Ж	ж	Ї	ї	Џ	џ
0470	Ψ	ψ	Θ	θ	ϐ	ϐ	ϑ	ϑ	ϒ	ϒ	Ϝ	Ϝ	ϝ	ϝ	Ϟ	Ϟ
0480	С	с	ѣ	ѣ	Є	є	А	а	І	і	Ж	ж	Ї	ї	Џ	џ
0490	Г	г	Ғ	ғ	Б	б	Ж	ж	З	з	Қ	қ	К	к	К	к
04A0	К	к	Ғ	ғ	Н	н	П	п	Q	q	С	с	Т	т	У	у
04B0	Ғ	ғ	Х	х	Ц	ц	Ч	ч	Ч	ч	Һ	һ	Е	е	Е	е
04C0	І	Џ	џ	К	к	Л	л	Н	н	Ғ	ғ	Ч	ч	М	м	І
04D0	Ӑ	ӑ	Ӓ	ӓ	Ӕ	ӕ	Ӗ	ӗ	Ә	ә	Ә	ә	Ӛ	ӛ	Ӝ	ӝ
04E0	Ӟ	ӟ	Ӡ	ӡ	Ӣ	ӣ	Ӥ	ӥ	Ӧ	ӧ	Ө	ө	Ӫ	ӫ	Ӭ	ӭ

Lire un fichier texte

Démo

```
file = open("movies.txt", "r", encoding="utf-8")  
contents = file.read()  
file.close()
```

En fait, on n'écrit pas ceci!

Pour être sûr de ne pas oublier le `close()`, on utilise un *with... as*

On ouvre ce fichier en lecture (*read*)

```
with open("movies.txt", "r", encoding="utf-8") as file:  
    contents = file.read()
```

```
print(contents)
```

Lecture!

On donne toujours l'encodage à utiliser pour ne pas avoir de surprise

Contient, sous forme de grand string unique, l'intégralité du fichier *movies.txt*

Écrire un fichier texte

Comme pour la lecture, on utilise un `with... as`, qui fait un `close()` automatique

On ouvre ce fichier en écriture (`write`)

```
with open("results.txt", "w", encoding="utf-8") as file:  
    file.write(...)
```

Le travail est fait par la méthode `write(...)`

```
file.write("une ligne\nune autre ligne")
```

Pour écrire plusieurs lignes, on doit indiquer le caractère de nouvelle ligne `\n`

```
for ....:  
    file.write(" ... \n")
```

Si les résultats à écrire sont générés au fur et à mesure, on peut les écrire petit à petit (ici, ligne par ligne) avec des appels répétés de `write()`

Traiter les données lues

- Pour de petits fichiers, on peut **tout lire d'un coup...**
- ... Mais comment traiter le contenu **ligne par ligne?**
 - Méthode `split()`, qui sépare une chaîne de caractère en une liste de sous-chaînes
 - **Conversions** de sous-chaînes en int, en datetime, en bool, etc.
 - **Transformations** de chaînes avec tests, `split()`, `replace()`
 - **Modélisation** des données lues avec des classes
- Exemple: lecture du fichier *movies.txt*

Démo

```
Year;Length;Title;Subject;Actor;Actress;Director;Popular
INT;INT;STRING;CAT;CAT;CAT;CAT;INT;BOOL;STRING
1990;111;Tie Me Up! Tie Me Down!;Comedy;Banderas, Antoni
1991;113;High Heels;Comedy;Bosé, Miguel;Abril, Victoria;
1983;104;Dead Zone, The;Horror;Walken, Christopher;Adams
1979;122;Cuba;Action;Connery, Sean;Adams, Brooke;Lester,
1978;94;Days of Heaven;Drama;Gere, Richard;Adams, Brooke
1983;140;Octopussy;Action;Moore, Roger;Adams, Maud;Glen,
1984;101;Target Eagle;Action;Connors, Chuck;Adams, Maud;
1989;99;American Angels: Baptism of Blood, The;Drama;Ber
1985;104;Subway;Drama;Lambert, Christopher;Adjani, Isabe
1990;149;Camille Claudel;Drama;Depardieu, Gérard;Adjani,
1982;188;Fanny and Alexander;Drama;Ahlstedt, Börje;Adolp
1982;117;Tragedy of a Ridiculous Man;Drama;Tognazzi, Ugo
1066;102;A Man & a Woman;Drama;Trintignant, Jean-Louis
```

Lecture et modélisation

```
from dataclasses import dataclass
from typing import List
```

```
with open("movies.txt", "r", encoding="utf-8") as file:
    contents = file.read()
```

Lecture de tout le fichier d'un coup

```
lines = contents.split("\n")
```

Séparation du contenu de chaque ligne,
à chaque apparition du caractère \n

```
@dataclass
class Movie:
    title: str
    year: int
    duration: int
    has_awards: bool
```

On décide d'une représentation sous de forme de
classe pour nos données à traiter

Création des objets depuis le texte lu

```
movies: List[Movie] = []
```

On prépare une liste de *Movies*, pour l'instant vide

```
substitutions = {  
    "&": "and",  
    "'": "'",  
    "vs.": "vs",  
}
```

Les substitutions de caractères qu'on voudra faire dans le titre des films

```
for line in lines[2:]:
```

On saute les deux premières lignes (cf. exemple de contenu)

```
    parts = line.split(";")
```

Chaque ligne est segmentée selon les points-virgules qu'elle contient

```
    duration_str = parts[1]
```

```
    if duration_str == "":
```

```
        duration = 0
```

```
    else:
```

```
        duration = int(duration_str)
```

De temps en temps, la durée est vide... on la met alors à 0. Sinon, on convertit de *str* vers *int*

```
    title = parts[2]
```

```
    if ", " in title:
```

```
        title = " ".join(title.split(", ")[:-1])
```

```
    for key, value in substitutions.items():
```

```
        title = title.replace(key, value)
```

On «nettoie» le titre, on applique les substitutions

```
    movie = Movie(title, int(parts[0]), duration, parts[8] == "Yes")
```

```
    movies.append(movie)
```

On crée un nouvel objet de type *Movie* et on l'ajoute à la liste

Traitement des données et écriture

```
movies.sort(key=lambda m: m.duration)
```

On trie les films selon un critère de tri donné par ce que cette fonction lambda retourne pour chaque film; ici, sa durée

On prépare un fichier de sortie pour y écrire

```
with open("results.txt", "w", encoding="utf-8") as file:  
    total_duration = 0
```

On y écrit ce qu'on veut: ici, la durée, le titre et l'année des films qui ont reçu un prix, chacun sur une ligne

```
for movie in movies:  
    if movie.has_awards:  
        file.write(f"{movie.duration:3} min: \"{movie.title}\" ({movie.year})\n")  
        total_duration += movie.duration
```

```
file.write(f"\nTotal duration: about {total_duration // 60} hours")
```

Dans la boucle, on en a profité pour calculer la durée totale de ces films, qu'on écrit comme ligne finale de ce fichier

Résumé: opérations sur les strings

- Tests

- Test de longueur, de casse
- Test de préfixe, suffixe, contenance

```
text = "..."
```

```
if len(text) > 20:  
if text == text.lower():
```

```
if text.startswith("..."):  
if text.endswith("..."):  
if "..." in text:
```

- Split

- Sépare un string en une liste selon un délimiteur

```
text = "a,b,c"  
parts = text.split(",")
```

- Join

- Crée un string à partir d'une liste en insérant un délimiteur

```
", ".join(parts)
```

- Replace

- Rechercher-remplacer

```
new_text = text.replace("a", "A")
```

Pas discuté: les expressions régulières, un moyen très puissant de manipuler du texte

Résumé Cours II

- Un caractère est représenté par une série de bytes selon un **encodage** précis
 - Aujourd'hui, **UTF-8** est le plus utilisé
- On peut facilement **lire et écrire** des fichiers texte en Python
 - Nouvelle structure: le **with... as** pour un *close()* automatique
- On peut **traiter** les données lues avec:
 - Des **transformations** de strings
 - Des **conversions** vers d'autres types
 - La construction d'objets via une **modélisation** par des classes