

Introduction à la Programmation : Mini-projet 1 en auto-évaluation

Rafael Pires
rafael.pires@epfl.ch

Objectifs du cours d'aujourd'hui

- Présentation générale du projet
- La représentation des données dans le projet
- La cryptographie et la stéganographie
- Mise en place du projet
- Les pièges communs et conseils

Stéganographie



Challenge : Jouez les détectives



Capture de drapeau : **FLAG{...}**

Prix aux 10 premiers groupes à atteindre 100 points



**But : atteindre au moins 60 points avant le 04.11.2024
(après cette date, le système d'évaluation sera fermé)**

Ce projet n'est pas noté.

On peut néanmoins mettre des questions
liées à ce projet dans l'examen.

Objectifs du projet

Coder un ensemble de méthodes permettant de :

- Cacher des messages textuels cryptés, ou des images, dans des images;
- Et de les dévoiler.

Objectifs pédagogiques :

- Auto-évaluer votre niveau sur un projet de plus grande envergure.
- Acquérir des premiers reflexes sur les méthodologies de tests d'un programme.
- Vous familiariser avec l'utilisation d'un debugger.
- Expérimenter le travail collaboratif et (optionnellement) l'utilisation d'un outil de gestion de version.
- Entraînement pour le 2^e mini-projet, qui sera noté et plus conséquent.

Conseils pour le travail collaboratif



- **Git : Outil de gestion de version.**
- Il permet de gérer un **dépot** (repository, en anglais) en fournissant les fonctionnalités suivantes :
 - Plusieurs personnes peuvent modifier et gérer en même temps les mêmes documents.
 - Enregistrement de l'historique de modification des documents.
 - Possibilité de revenir en arrière à une version précédente quelconque.

[Ressources.](#)

Objectifs du cours d'aujourd'hui

- Présentation générale du projet
- La représentation des données dans le projet
- La cryptographie et la stéganographie
- Mise en place du projet
- Les pièges communs et conseils

Représentation des données : Texte

UTF-8 : Unicode Transformation Format – 8-bit.

Norme d'encodage de caractères la plus utilisée aujourd'hui.

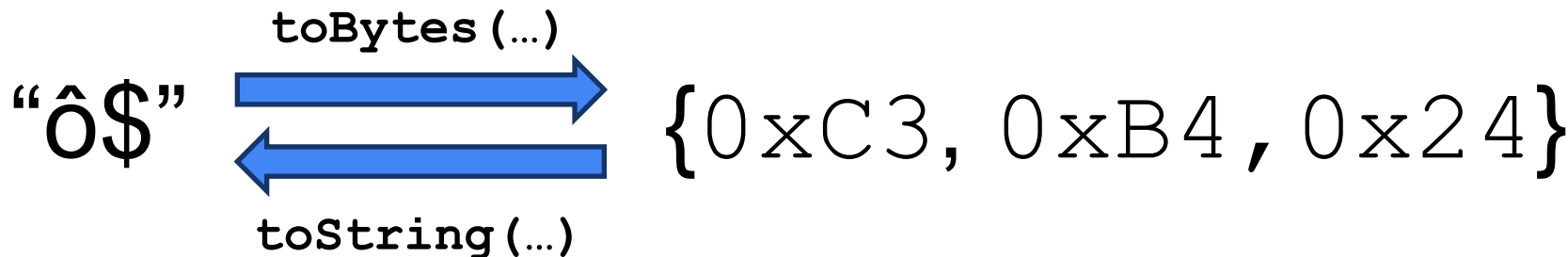
- Capable d'encoder plus d'un million de points de code Unicode valides.
- Encodage de largeur variable, allant de un à quatre octets.
- Les points de code les plus fréquents sont encodés utilisant moins d'octets.
- Rétrocompatible avec l'ASCII : les 128 premiers caractères (1 octet).

“ô” $\rightarrow \{0xC3, 0xB4\}$

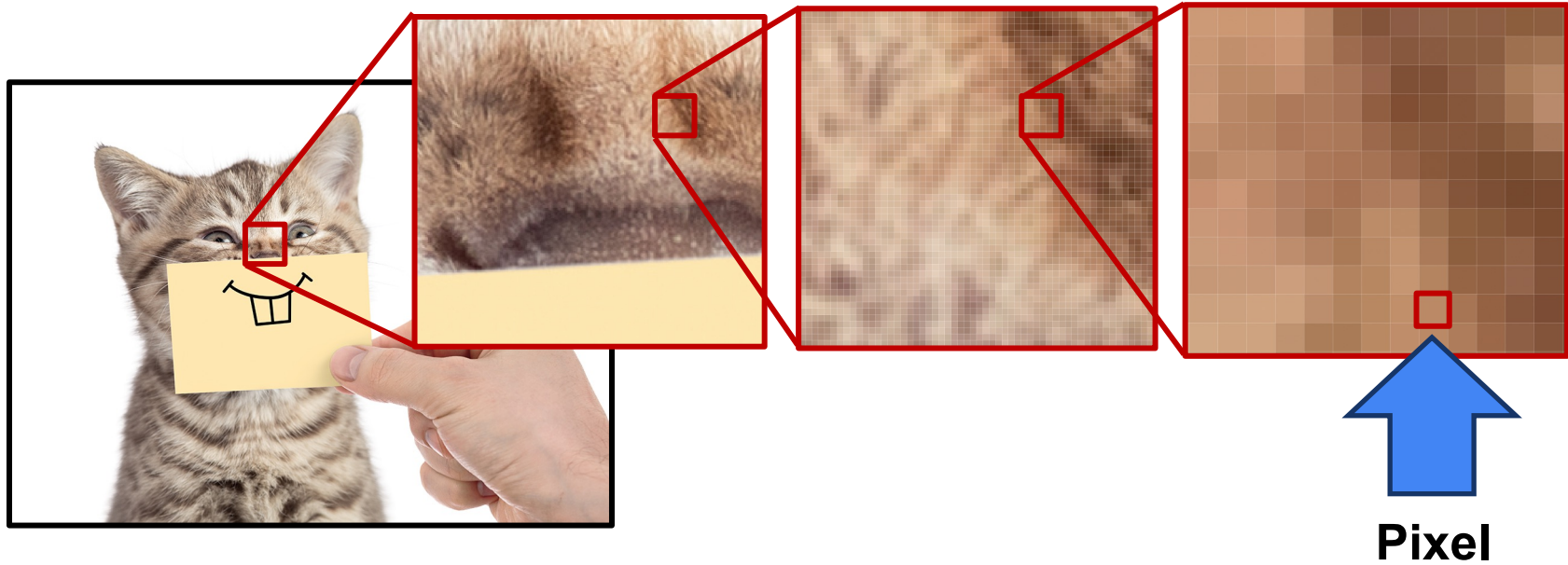
“\$” $\rightarrow \{0x24\}$

Les chaînes de caractères en Java

- En Java, les chaînes de caractères sont représentées par le type **String**, mais leur encodage peut varier d'une machine à l'autre.
- Dans ce projet, nous manipulerons les chaînes de caractères sous un format plus fondamental, les **tableaux de byte**, avec un encodage fixé à **UTF-8**.
- Pas de panique : on vous fournit les méthodes suivantes dans **Text.java**.



Représentation des données : Images



Pixel

Point de l'image qui a une seule couleur

ARGB

Alpha



Transparency

Red



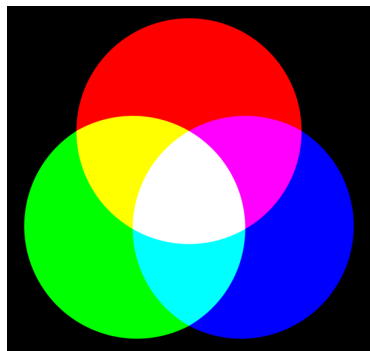
Green



Blue



Couleur



Noir = (0, 0, 0)

Blanc = (255, 255, 255)

Bleu = (0, 0, 255)

1 pixel = 32 bits = 4 x 8 bits = 4 nombres entre 0-255

Objectifs du cours d'aujourd'hui

- Présentation générale du projet
- La représentation des données dans le projet
- La cryptographie et la stéganographie
- Mise en place du projet
- Les pièges communs et conseils

Cryptographie

La science de sécuriser des informations en les transformant de manière à les rendre illisibles pour toute personne non autorisée, grâce à des techniques de **chiffrement** et de **déchiffrement**.

Vocabulaire :

- **Message en clair (*plaintext*)** : message à chiffrer.
- **Message chiffré/crypté (*cipher text*)** : version chiffrée du message.
- **Crypto-système (*cipher*)** : algorithme permettant de chiffrer/déchiffrer un message.
- **Clé (*key*)** : information secrète utilisée pour chiffrer/déchiffrer un message.

Cryptographie

Crypto-systèmes à implémenter :

- César
- Vigenère
- XOR
- OTP (One-time pad)
- CBC (Cipher block chain)

Les descriptifs détaillés sont dans l'énoncé, avec des exemples d'exécution attendue dans le fichier fourni **Main.java**.

Cryptographie

Important : le but est de chiffrer des textes exprimés comme tableaux de **byte**.

L'alphabet utilisé est donc l'ensemble des **bytes** possibles : toutes les valeurs entre **-128** et **127**.

Exemple : **chiffrement de César**

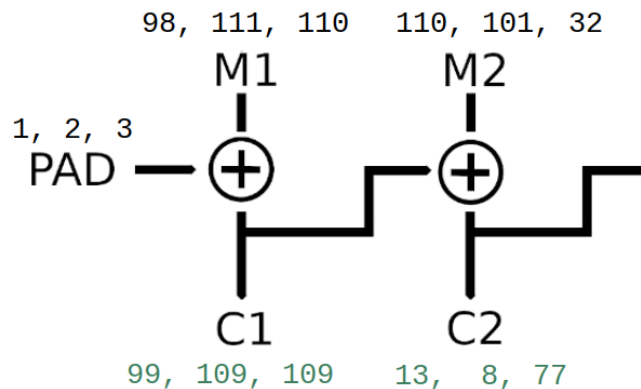
Chaque **byte** de la chaîne à chiffrer est décalé de la valeur d'une clé donnée (aussi un **byte**), modulo la taille de l'alphabet.

- si le byte vaut 97 et la clé 3, le byte encrypté vaudra 100
- décrypter revient à crypter avec la clé inverse, -3

Crypto-système CBC (simplifiée)

- Décompose le message en blocs de la taille du pad.
- Effectue un XOR entre les blocs et le pad.

```
byte[] plain  = {98, 111, 110, 110, 101, 32};  
byte[] pad    = {1, 2, 3};  
byte[] cipher = Encrypt.cbc(plain, pad); // {99, 109, 109, 13, 8, 77}
```



XOR		
0	0	0
0	1	1
1	0	1
1	1	0

Stéganographie

Rouge: 01100100

Vert: 11001000



Bleu : 11111111



Bleu : 11111110



LSB = Least significant bit



Bleu : 11110000



Bleu : 11000000



Changer le LSB change très faiblement la couleur.



On va utiliser le LSB de chaque pixel pour y stocker un message secret.

Stéganographie : images

- **But** : cacher une image en noir et blanc dans les LSB des pixels.



- La taille de l'image cachée (charge) doit être inférieure ou égale en hauteur et en largeur à celle de l'image de couverture.
- Les positions des pixels de la charge sont maintenues.

Stéganographie : messages textuels

- **But** : cacher un texte représenté par une séquence de bytes linéairement dans les LSB des pixels.



"Hello" $\rightarrow$ {72, 101, 108, 108, 111} $\rightarrow$
{01001000, 01100101, 01101100, 01101100, 01101111}

Objectifs du cours d'aujourd'hui

- Présentation générale du projet
- La représentation des données dans le projet
- La cryptographie et la stéganographie
- Mise en place du projet
- Les pièges communs et conseils

Enoncé et mise en place

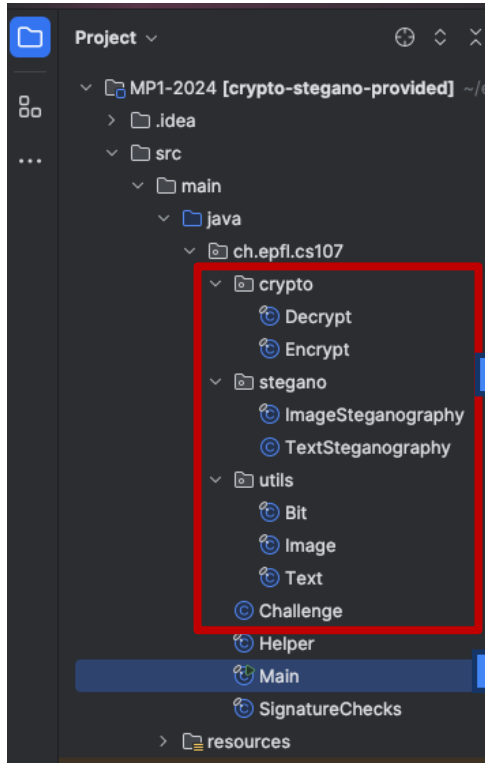
- **Enoncé** : Liens disponibles sur [cette page](#).



- **Mise en place** : Petit tutoriel sur [cette page](#).



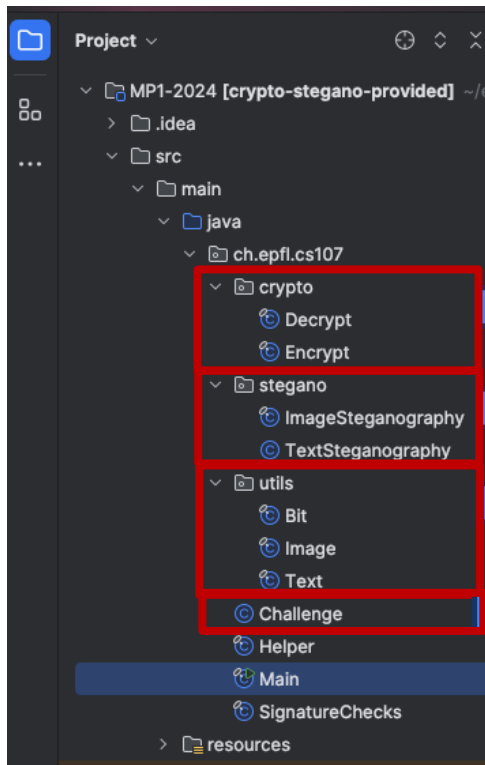
Matériel fourni



À compléter : tous les fichiers des dossiers **utils**, **crypto** et **stegano** et **Challenge.java**. Entêtes fournis.

À lancer : **Main.java**. À compléter pour bien tester !

Tâches



➤ Les tâches 2 et 3 peuvent être codées **indépendamment**.

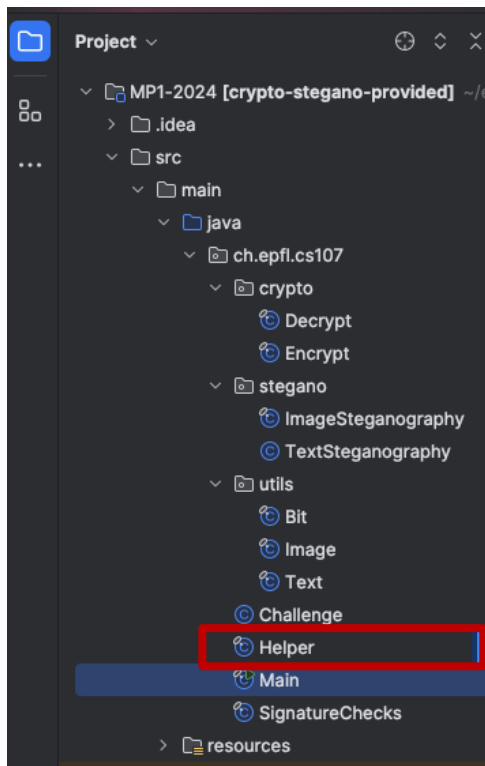
Tâche 2 : Méthodes de cryptographie sous **crypto**.

Tâche 3 : Méthodes de steganographie sous **stegano**.

Tâche 1 : Méthodes utilitaires sous **utils**.

Tâche 4 : Challenge sous **Challenge.java**.

Appels aux méthodes dans un autre fichier



- Syntaxe : **NomDeClasse.nomDeMethode (...)**
- Exemple : `Helper.read("image.png")`

N'oubliez pas de regarder **Helper.java**.

Correcteur automatique



- Bien tester localement avant de soumettre.
- Ne soumettre que du code qui compile.
- Faire attention aux cas limites (voir le mécanisme des assertions dans l'énoncé).
- Ne jamais utiliser d'appels système comme `System.exit()`.
- Préparez le fichier “`submission.zip`” d'après les [consignes](#).

Merci de nous alerter sur Ed en cas de souci avec l'utilisation de cet outil !

Les entiers en Java (1/3)

Tous les entiers en Java sont signés !

Nous les utilisons le plus souvent en base 10 .

➤ il ne s'agit que d'une représentation

Trois façons d'écrire la même chose :

13 → $3*10^0 + 1*10^1$

0b1101 → $1*2^0 + 0*2^1 + 1*2^2 + 1*2^3$

0xD → $D*16^0$

(en base 16, A vaut 10, B vaut 11, ... F vaut 15)

un byte est codé sur un octet (8 bits)

➤ Entier signé valant entre -128 et 127

Les entiers en Java (2/3)



Arithmétique en **complément à deux** (voir [la vidéo](#) fournie)

Les opérations **+** et **-** se font modulo les plus grand/petit nombres représentables :

- Pas de débordement
 - Si un **byte** vaut **127**, lui additionner **1** donnera **-128**
 - Si un **byte** vaut **-128** lui retrancher **1** donnera **127**

Soyez attentifs à la promotion entière : les calculs sur des **byte** renvoient toujours un **int**

- **transtypage nécessaire**

```
byte b = 10;  
byte sum = b + b;           // compile error  
byte sum = (byte) (b + b);  // cast necessary
```

Les entiers en Java (3/3)

Un `int` est représenté sur 4 octets (32 bits).

Transtyper un `int` en `byte` tronque les 3 octets les plus à gauche.

Transtyper un `byte` en `int` remplit le 3 octets les plus à gauche de 1 si le `byte` est négatif et de 0 sinon (complément à deux).

```
byte b = 127;           // 0b01111111
int i = 127 + 1;        // 0b00000000_00000000_00000000_10000000 = 128
byte b2 = (byte) i;     // 0b00000000_00000000_0000000010000000 = -128
int i2 = (byte)-128;    // 0b11111111_11111111_11111111_10000000 = -128
```

Utilisez le debugger pour examiner le contenu des variables (représentation binaire)

Les opérateurs binaires **&**, **|**, **^**, **~**, **<<**, **>>**, **>>>**

Ils s'appliquent à la représentation binaire des nombres.

Ils vont être utilisés de façon omniprésente :

- par certains algorithmes de **chiffrement**
- pour **accéder à un bit particulier** dans le but d'y stocker de l'information
- pour coder des fonctionnalités utiles sur les **images**, par exemple :
 - extraction des canaux de couleurs.
 - produire une représentation en niveau de gris d'une image.

Voir la documentation de Java pour une liste exhaustive de ces opérateurs.

Les opérateurs binaires : << et |

```
int val1 = 1 << 7;
```



```
// 0b10000000
```

```
int val2 = val1 | 1;
```

```
// 0b10000001
```

OR		
0	0	0
0	1	1
1	0	1
1	1	1

Les opérateurs binaires : << et |

```
int val1 = 0b10;  
int val2 = 0b10001;
```

But : Obtenir 0b10010001.

1. Shift left avec insertion de 0s à droite (<<)

```
int val1_s1 = val1 << 6;  
// 0b10000000
```

2. Bitwise OR

```
int val3 = val1_s1 | val2;  
// 0b10010001
```

Les opérateurs binaires : $\ggg$, $\gg$ et $\&$

```
byte val1 = -128;  
byte val2 = val1  $\gg$  7;
```

```
// 0b10000000  
// 0b11111111 = -1  
7x1 
```

Décalage arithmétique
conserve le signe

```
byte val3 = val1  $\ggg$  7;
```

```
// 0b00000001 = 1  
7x0 
```

Décalage logique
ne conserve pas le signe

```
int val2 = val2  $\&$  0b11;  
// 0b00000011
```

AND		
0	0	0
0	1	0
1	0	0
1	1	1

Les opérateurs binaires : >>>, >> et &

```
int val1 = 0b10010001;
```

But : Extraire les bits rouges et bleus séparément.

1. Shift right avec insertion de 0s à gauche (>>>)

```
int val1_sr = val1 >>> 6;  
// 0b10
```

2. Bitwise AND avec une masque

```
int val2 = val1 & 0b11111;  
// 0b10001
```

>>> insertion de 0s à gauche.

>> insertion de 0s pour nombres positifs et de 1s pour les négatifs sur la gauche.

Objectifs du cours d'aujourd'hui

- Présentation générale du projet
- La représentation des données dans le projet
- La cryptographie et la stéganographie
- Mise en place du projet
- Les pièges communs et conseils

Les pièges



1. Ordre de bits.
2. Chiffres positifs / négatifs.
3. Cas limites.

Piège 1 : Ordre de bits



On vous demande de transformer un octet en un tableau de booléens, et vice-versa.
Par exemple :

Indices aux opérateurs binaires

`<<`, `>>>` et `>>` : 7654**3**210.



droite à gauche



```
byte value = 0b00110110;  
boolean[] bits = toBitArray(value);  
// bits = {false, false, true, true, false, true, true, false}
```

Indices aux tableaux : 0 1 2 **3** 4 5 6 7



gauche à droite



Piège 2 : Nombres positifs / négatifs

Comment faire de l'arithmétique non signée entre des nombres de type `byte` ?

```
byte b = 0b10010000;           // 144 (non signé) = -112 (signé)  
System.out.println(b / 12);    // affiche -9
```

Comment on fait pour que le résultat soit 12 ?

```
System.out.println(Byte.toUnsignedInt(b) / 12); // affiche 12
```



Piège 3 : Cas limites

Il faudra s'assurer que les paramètres des méthodes sont corrects en utilisant ce que l'on appelle des **assertions**.

```
public static int[][] embedBW(int[][] cover, boolean[][] image)
{
    assert (cover != null);
    assert (cover.length >= image.length);
    //...
}
```


Conseils pour aborder le mini-projet 1

- Commencez par lire l'introduction, les consignes et les compléments théoriques.
- Lisez le descriptif des tâches avec soin avant d'entamer le codage.
- Testez systématiquement en local, avant de soumettre au correcteur automatique.
- N'hésitez pas à enrichir le fichier **Main.java** fourni pour pousser les tests plus loin.
- Ne tentez pas de résoudre tous les problèmes en même temps.
- Ne négligez pas les apprentissages et séries à venir au profit de ce projet.

Essayez [ces exercices](#) avant le TP de vendredi.



Questions ?



Nous sommes à disposition sur le forum **Ed** pendant la semaine.
N'hésitez pas à poser des questions.

Bon projet à toutes et tous !

rafael.pires@epfl.ch



EPFL

Merci