Algorithmique & Programmation

De l'impératif à l'objet

yann.secq@univ-lille.fr

ALMEIDA COCO Amadeu, BIRLOUEZ Martin, BONEVA Iovka, DELECROIX Fabien, LEPRETRE Éric, MARSHALL-BRETON Christopher, ROUZÉ Timothé, SECQ Yann, SOW Younoussa, SUE Yue

Saison 2024-2025 / R1.01+R1.02
Initiation au développement

De l'impératif à l'objet

- Initiation à la programmation impérative
- Appropriation des concepts fondamentaux de l'algorithmique : variable et affectation, séquence, répétition et alternative, fonction
- Comment organiser un logiciel complexe impliquant des dizaines / centaines / milliers de lignes ?
- Notion de fonction, notion de module (cf. extensions)

De l'impératif à l'objet

- Notion de type introduite en S1
- Permet de caractériser précisément la nature des informations manipulées
- Permet des vérifications par le compilateur
- **Crée un langage propre au problème modélisé (avec le nommage des variables et des fonctions !)**

Objet : données/état + traitements

- En POO, on modélise différemment en regroupant les données et leurs traitements associés dans une même entité
- Un **objet** est constituée de **données** (état) et de **traitements** (comportements)
- Un objet défini un type, mais avec les comportement en plus des états présents dans les types que nous utilisons actuellement

Ex. simple : un compteur

- Un type compteur défini par une borne de début, une borne de fin et un pas d'incrémentation
- Une opération permettant de connaître la valeur courante du compteur
- Une opération permettant de passer à la prochaine valeur

ÉTAT

COMPOR
TEMENTS

```
class Compteur {

  int valeur;
  final int increment;
  final int borne;

  Compteur(int borneDebut, int borneFin, int increment) {
    valeur = borneDebut;
    this.increment = increment;
    borne = borneFin;
  }
  Compteur(int borneDebut, int borneFin) { // +1 par défaut
    this(borneDebut, borneFin, 1);
  }

  boolean increment() {
    boolean incrementPossible = valeur+increment < borneFin
    if (incrementPossible) {
      valeur = valeur + increment;
    }
    return incrementPossible;
  }

  int etat() {
    return valeur;
  }
}
```

ÉTAT

COMPOR
TEMENTS

```
class Compteur {

  int valeur;
  final int increment;
  final int borne;

  Compteur(int borneDebut, int borneFin, int increment) {
    valeur = borneDebut;
    this.increment = increment;
    borne = borneFin;
  }
  Compteur(int borneDebut, int borneFin) { // +1 par défaut
    this(borneDebut, borneFin, 1);
  }

  boolean increment() {
    boolean incrementPossible = valeur+increment < borneFin
    if (incrementPossible) {
      valeur = valeur + increment;
    }
    return incrementPossible;
  }

  int etat() {
    return valeur;
  }
}
```

```
class UsageCompteur {
  public static void main(String[] args) { // == void algorithm() !
    Compteur cpt1 = new Compteur(0, 10, 2);
    Compteur cpt2 = new Compteur(0, 5);
    for (int i=0; i<3; i++) {
      cpt1.increment();
      cpt2.increment();
    }
    System.out.println(cpt1.etat()+" / "+cpt2.etat());
  }
}
```

Enrichissement de notre langage pour modéliser

- On dispose maintenant de la notion de compteur que l'on peut manipuler sans avoir à connaître son implémentation
- C'est la notion d'encapsulation
- C'est ce qui se passe presque lorsque vous utilisez les « extensions » de *ijava*

Taxonomie de concepts

- Parfois des types sont proches : ils partagent des états ou des comportements
- Comment éviter la duplication de code et d'information ?
- Usage de la notion d'héritage !

Taxonomie de concepts

- Une **personne** : nom, prénom, date de naissance, code INSEE
- Un·e **étudiant·e** : un code étudiant, une formation, un groupe
- Un·e **enseignant·e** : un code personnel, une liste de cours

```
class Personne {
  String nom, prenom;
  final Date naissance;
  final Code insee;

  int age() {
    return Date.aujourdhui().moins(naissance)
  }
}
```

```
class Etudiant extends Personne {  // Etudiant est une personne particulière
  final Code nip;
  Formation formation;
  Groupe groupe;
  //…
}
```

```
class Enseignant extends Personne { // Enseignant est une personne particulière
  final Code personnel;
  List<Cour> cours;
  // …
}
```

```
class UsagePersonnes {
  public static void main() {
    List<Personne> iut = // initialisation avec des étudiants et enseignant

    for (Personne p : iut) {
      System.out.println(p.age()); // polymorphisme !
    }
  }
}
```

Voir qqchose selon un certain point de vue ?

- Parfois on souhaite faire un même traitement avec un critère différent sur un certain type
- Ex: **trier** des personnes **selon leur nom**, ou prénom ou **selon leur âge** …
- Notion d'interface / de contrat :)

```
class Personne {
  String nom, prenom;
  final Date naissance;
  final Code insee;

  int age() {
    return Date.aujourdhui().moins(naissance)
  }
}
```

```
interface Comparateur {
   int compare(Personne p1, Personne p2);
}
```

```
class TriSurAge implements Comparateur {
   int compare(Personne p1, Personne p2) {
     return p1.age()-p2.age();
   }
}
```

```
class TriSurNom implements Comparateur {
   int compare(Personne p1, Personne p2) {
     return p1.nom.compare(p2.nom);
   }
}
```

```
class UsagePersonnes {
  public static void main() {
    List<Personne> iut = // initialisation avec des étudiants et enseignant
    Collections.sort(iut, new TriSurAge());
    afficher(iut);
    Collections.sort(iut, new TriSurNom());
    afficher(iut);
  }
}
```

De l'impératif à l'objet

- Bonne nouvelle : toute l'algorithmique reste valable et est mobilisée en POO :)
- Peu de concepts en POO, comme en impératif, mais important de bien les comprendre : objet, classe, attributs, méthodes, encapsulation, héritage, interface
- Nécessite de la pratique pour s'habituer à modéliser de manière orientée objets
- Programme du S2 avant d'aller plus loin au S3 !

Algorithmique & Programmation

Extensions iJava
Programmation évènementielle & graphisme

yann.secq@univ-lille.fr

ALMEIDA COCO Amadeu, BIRLOUEZ Martin, BONEVA Iovka, DELECROIX Fabien, LEPRETRE Éric, MARSHALL-BRETON Christopher, ROUZÉ Timothé, SECQ Yann, SOW Younoussa, SUE Yue

Saison 2024-2025 / R1.01+R1.02
Initiation au développement

Extensions iJava

- Avant tout: ce n'est pas si simple !
- ASCII Art en couleurs
- Graphisme simple avec le type `Image`
- Sinon, il y a aussi `Graphism` ... ;)
- Préalable: récapitulatif sur `Program`

Mode texte

void	**background**(java.lang.String color) — Défini la couleur de l'affichage du fond du texte.
void	**backward**() — Déplace le curseur vers la gauche.
void	**backward**(int n) — Déplace le curseur de n cases vers la gauche.
void	**clearBOL**() — Efface la ligne depuis le début de la ligne jusqu'au curseur.
void	**clearEOL**() — Efface la ligne depuis la position jusqu'à la fin de la ligne.
void	**clearLine**() — Efface l'ensemble du contenu de la ligne courante.
void	**clearScreen**() — Efface l'ensemble de l'écran et repositionne le curseur en (1, 1).
void	**curp**() — Restaure la position sauvegardée du curseur (CUrsor Restore Position).
void	**cursor**(int line, int column) — Positionne le curseur aux coordonnées (line, column).
void	**cusp**() — Sauvegarde la position courante du curseur (CUrsor Save Position).
void	**down**() — Déplace le curseur vers le haut.
void	**down**(int n) — Déplace le curseur de n lignes vers le bas.
void	**enableKeyTypedInConsole**(boolean on) — Active (true) ou désactive (false) la possibilité de capturer les touches manipulées par l'utilisateur.
void	**forward**() — Déplace le curseur vers la droite.
void	**forward**(int n) — Déplace le curseur de n cases vers la droite.
void	**hide**() — Masque le curseur.
java.lang.String	**randomANSIColor**() Cette fonction retourne aléatoirement une chaîne de caractère représentant une couleur (parmi les couleurs définies dans le
void	**reset**() — Réinitialise la console (utile lorsque vous jouez avec les couleurs du texte !).
void	**show**() — Affiche le curseur.
void	**text**(java.lang.String color) — Défini la couleur de l'affichage du texte.
void	**up**() — Déplace le curseur vers le haut.
void	**up**(int n) — Déplace le curseur de n lignes vers le haut.

Capture de touche

- Capter lorsque l'utilisateur appuie sur une touche
- Notion d'évènement: « *la touche* `'a'` *a été utilisée par l'utilisateur* »
- Une méthode spécifique est appelée si cela se produit
- D'abord basculer en « écoute » des touches puis définir: `void keyTypedInConsole(char key)`

Fonction appelée automatiquement dès que l'utilisateur appuie sur une touche

```
// Tant que ce booléen est à faux
// le programme s'exécute
boolean finished = false;


void algorithm() {
  // On active l'écoute des évènements clavier
  enableKeyTypedInConsole(true);
  while (!finished) {
    delay(500);
  }
}

// Fonction définissant ce qui doit être fait
// lorsqu'une touche est pressée par l'utilisateur
void keyTypedInConsole(char key) {
  println("Vous avez appuyé sur : " + key
          + " (pressez 'q' pour quitter)");
  switch (key) {
    // ANSI_UP est une constante correspondant
    // au code ASCII de la flèche HAUT
    case ANSI_UP:
      println("Flèche UP !");
      break;
    // …
    case 'q' :
      println("Ok, au revoir ...");
      finished = true;
    }
}
```

```
class ManipulationConsole extends Program {
   boolean finished = false;
   void algorithm() {
        enableKeyTypedInConsole(true);
        while (!finished) {
          delay(500);
        }
   }
   void keyTypedInConsole(char key) {
        println("Vous avez appuyé sur : " + key +
                " (pressez 'q' pour quitter)");
        switch (key) {
            case ANSI_UP:
                println("Flèche HAUT !");
                break;
            case ANSI_DOWN:
                println("Flèche BAS !");
                break;
            case ANSI_LEFT:
                println("Flèche GAUCHE !");
                break;
            case ANSI_RIGHT:
                println("Flèche DROITE !");
                break;
            case 'q' :
                println("Ok, aurevoir ...");
                finished = true;
        }
    }
}
```

Le type `Image`

- Le type `Image` permet d'afficher une image existante ou d'en créer une nouvelle
- De nombreuses primitives permettent de créer, modifier et interagir avec une image
- Deux modes de dessin : totalement manuel ou en mode grille
- Comme les autres types, vous pouvez créer des tableaux d'images si nécessaire

Image: création/mode grille

Création

```
Image newImage(String name, String filename)
Image newImage(String filename)
Image newImage(String name, int width, int height)
Image newImage(int width, int height)
```

Visibilité

```
void show(Image img)
void hide(Image img)
void close(Image img)
```

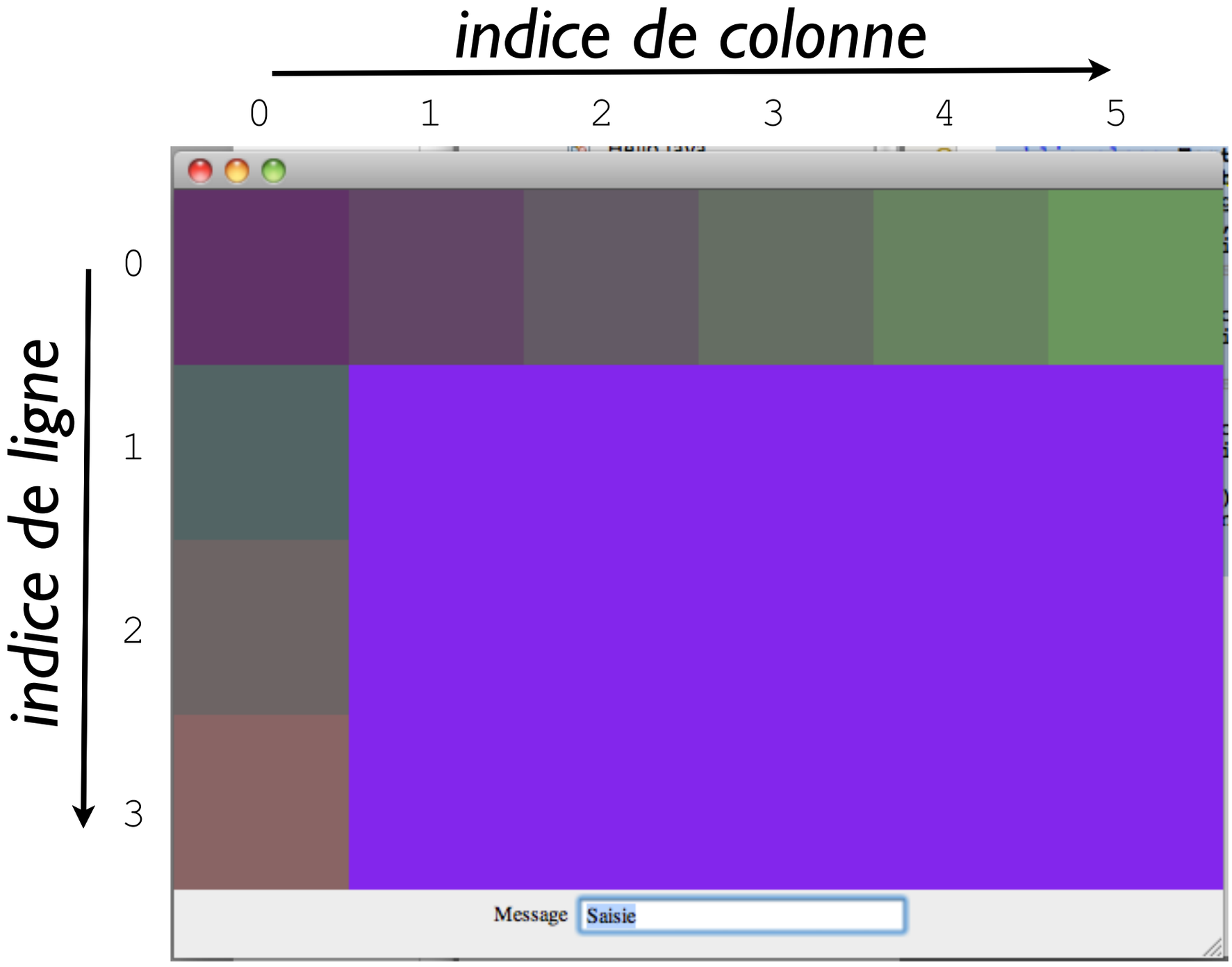
Couleurs

```
RGBColor randomColor()
int rgb(int r, int g, int b)
setColor(Image img, RGBColor color)
setColor(Image img, int r, int g, int b)
```

Mode grille

```
void setGrid(Image img, int lines, int columns)
int getGridLines(Image img) | int getGridColumns(Image img)
int get(Image img, int line, int column)
void set(Image img, int line, int column)
void set(Image img, int line, int column)
void set(Image img, int line, int column, int color)
void set(Image img, int line, int column, RGBColor color)
```

Exemple en mode grille



Exemple en mode grille

```
import extensions.Image;
import extensions.RGBColor;
class TestGrid extends Program {
  void algorithm() {
    Image img = newImage(600, 400);
    fill(img, RGBColor.BLUEVIOLET);
    setGrid(img, 4, 6);
    // Sur les lignes
    for (int line=0; line<getGridLines(img); line++) {
      setColor(img, (50+(line*30))%256, 100, 100);
      set(img, line, 0);
    }
    // Sur les colonnes
    for (int column=0; column<getGridColumns(img); column++) {
      set(img, 0, column, rgb((50+column*20)%256, 100, 100));
    }
    show(img);
    readString();
  }
}
```

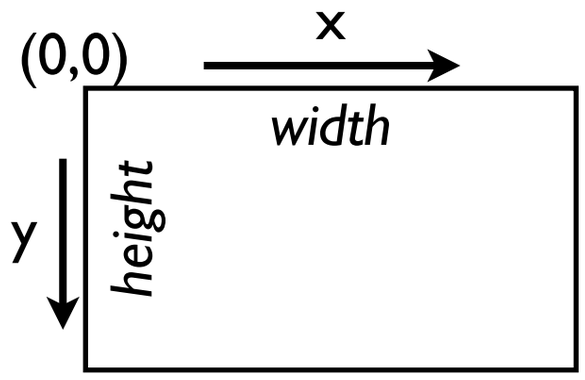
Le type `Image`

- Une `Image` est caractérisée par une largeur (*width*) et une hauteur (*height*)
- L'origine se trouve dans le coin supérieur gauche
- Une couleur est définie et utilisée dans les primitives de dessin
- Trois formes : segment, rectangle et ellipse
- Deux modes : contour ou remplissage

Image: **primitives**

Formes géométriques

```
void fill(Image img)
void fill(Image img, int color)
void fill(Image img, RGBColor color)
void drawLine(Image img, int x1, int y1, int x2, int y2)
void drawRect(Image img, int x, int y, int w, int h)
void fillRect(Image img, int x, int y, int w, int h)
void drawOval(Image img, int x, int y, int w, int h)
void fillOval(Image img, int x, int y, int w, int h)
```




```
int getWidth(Image img)
int getHeight(Image img)
```

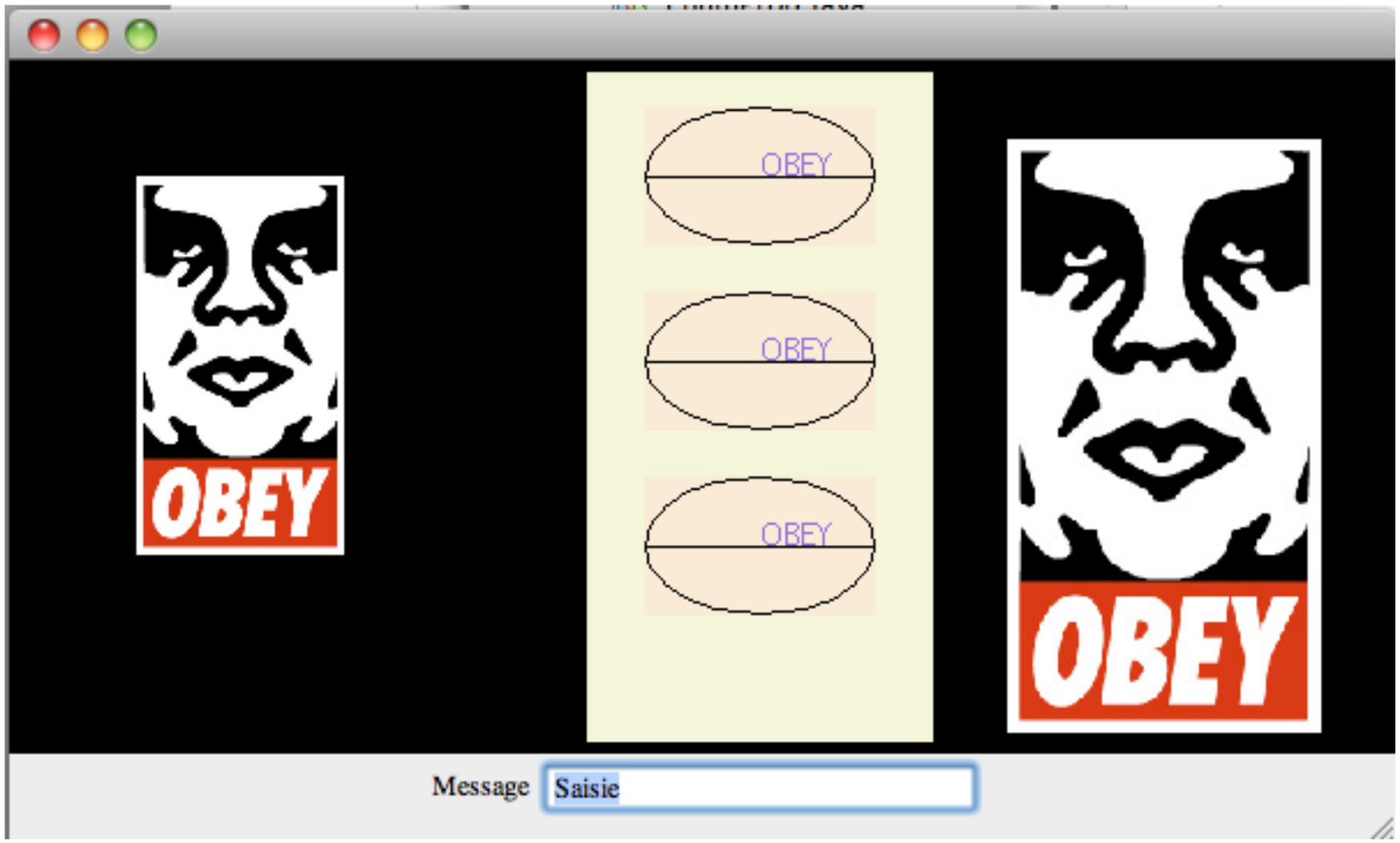
Texte et police

```
String[] getAllFontNames() {
void setNewFont(Image img, String name, String kind, int size)
void drawString(Image img, String text, int x, int y)
```

Manipulation d'image

```
void drawImage(Image img, Image pip, int x, int y)
Image copyAndResize(Image img, String name, int width, int height)
```

Dessinons manuellement



Dessinons manuellement

```
import extensions.*;
public class TestDrawImage extends Program {
  void algorithm() {
    Image img = newImage(600, 300);
    setColor(img, RGBColor.BEIGE);
    fillRect(img, 250, 5, 150, 290);

    Image original = newImage("Obey", "/Users/secq/Desktop/obey.jpg");
    drawImage(img, copyAndResize(original, "obey-small", 100, 175), 50, 45);
    drawImage(img, copyAndResize(original, "obey-medium", 150, 275), 425, 25);

    Image pip = newImage("Dessin", 100, 60);
    fill(pip,RGBColor.ANTIQUEWHITE);
    setColor(pip, RGBColor.BLACK);
    drawOval(pip, 0, 0, getWidth(pip)-1, getHeight(pip)-1);
    drawLine(pip, 0, getHeight(pip)/2, getWidth(img), getHeight(pip)/2);
    setColor(pip, RGBColor.MEDIUMPURPLE);
    setNewFont(img, "HELVETICA", "ITALIC", 32);
    drawString(pip,"OBEY", getWidth(pip)/2, getHeight(pip)/2);

    int cpt = 1;
    for (int i = 10; i < 200; i += 80) {
      setName(pip, "Dessin"+(cpt++));
      drawImage(img, pip, 275, 10 + i);
    }
    show(img);
    readString();
  }
}
```

Gestion de l'interactivité

```
void setMessage(Image img, String label)
void setText(Image img, String text)
```

- Deux modes d'interaction
  - **la zone de contrôle** : un texte, une zone de saisie et des boutons (en bas de la fenêtre)
  - **la zone du dessin** : touches du clavier et souris
- Pour gérer une action de l'utilisateur il faut écrire une procédure spécifique

Notion d'événement

- Une action de l'utilisateur est appelé un "événement"
- Types d'événements : activation d'un bouton, saisie de texte, déplacement ou clic de souris
- Chaque type d'événement est traité par un sous-programme spécifique (à définir !)

Gérer un bouton

```
void addButton(Image img, String label)
void changeButton(Image img, String oldlabel, String newlabel)
```

- Étapes nécessaires pour gérer un bouton
  - ajouter un bouton à l'interface : `addButton`
  - puis définir : `void buttonPushed(String buttonLabel)`
- Dès que l'utilisateur "clic" `buttonPushed` est appelé

```
import extensions.*;
class TestGrilleInteraction extends Program {
  final String CLEAR = "Clear";
  Image grille;

  int rnd() {
    return (int) (random() * 65535);
  }
  void algorithm() {
    grille = newImage(500, 300);
    addButton(grille, CLEAR);
    show(grille);
    setGrid(grille, 4, 7);
    for (int i = 0; i < 100; i++) {
      set(grille, rnd() % 4, rnd() % 7, randomColor());
      delay(250);
    }
    readString();
  }
  void buttonPushed(String buttonLabel) {
    println("Un bouton a été déclenché: " + buttonLabel);
    if (equals(buttonLabel, CLEAR)) {
      fill(grille, randomColor());
    }
  }
}
```

Gérer le clavier

- Soit au niveau de la zone de graphique :
  - détection fine des touches du clavier
    - définir : `void keyChanged(char c, String event)`
- Soit au niveau de la zone de saisie :
  - saisie d'une phrase
    - définir : `void textEntered(String text)`

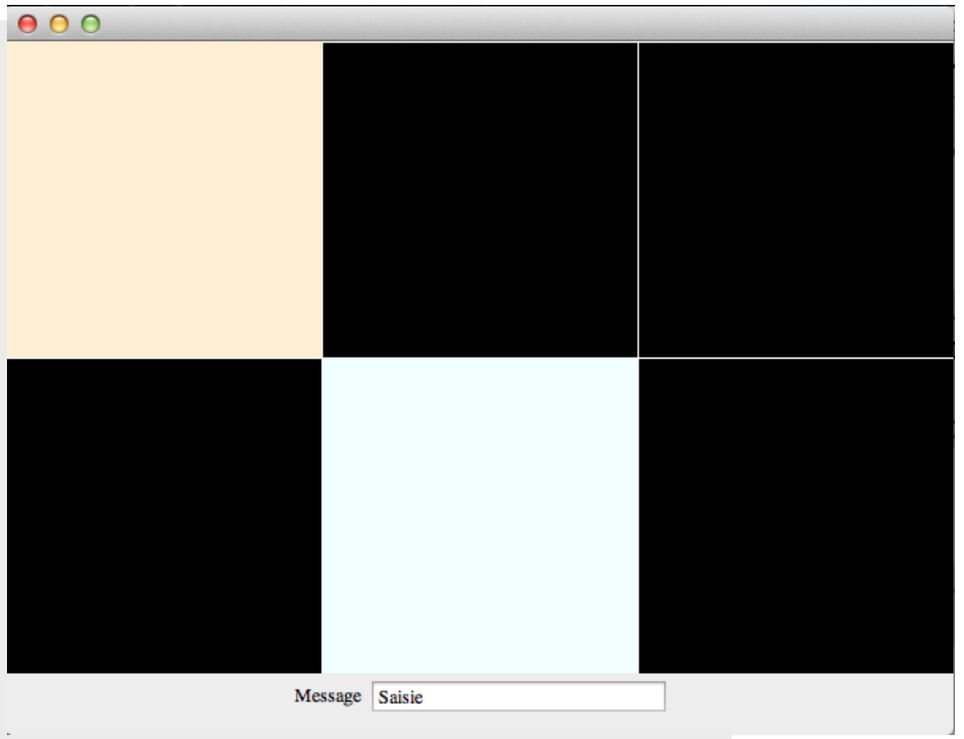
```
import extensions.*;
class TestInteractionClavier extends Program {

  void algorithm() {
    Image grille = newImage(500, 300);
    show(grille);
    readString();
  }

  // Gestion de la zone de saisie de texte
  void textEntered(String text) {
    println("Phrase saisie : " + text);
  }

  // Gestion des touches au niveau de la zone graphique
  void keyChanged(char c, String event) {
    println("Caractère saisi : " + c);
  }

}
```

Image: interactivité !

```
void addZone(Image img, String name, int x, int y, int w, int h)
void removeZone(Image img, String name)
```

- Définition de “zones” sensibles à la souris (entrée, sortie ou clic)
- Fait automatiquement pour les images
- Manuellement avec `addZone`
- Le nom donné à l’image ou à la zone est celui qui apparaîtra dans l’événement

```
import extensions.*;
class TicTacToeGUI extends Program {
  Image img;
  void algorithm() {
    final int LIGNES = 2, COLONNES = 3;
    final int HAUTEUR = 400, LARGEUR = 600;
    int hauteurZone = HAUTEUR/LIGNES;
    int largeurZone = LARGEUR/COLONNES;
    img = newImage(LARGEUR, HAUTEUR);
    setGrid(img, LIGNES, COLONNES);
    show(img);
    for (int lig=0; lig<LIGNES; lig++) {
      for (int col=0; col<COLONNES; col++) {
        int origineX = col*largeurZone;
        int origineY = lig*hauteurZone;
        addZone(img, "("+lig+","+col+")", origineX, origineY,
          largeurZone, hauteurZone);
        drawRect(img, origineX, origineY,
          largeurZone, hauteurZone);
        }
    }
    readString();
  }
  void mouseChanged(String name, int x, int y, int button, String
event) {
    println("Zone "+name+" a genere l'evenement "+event);
    if (equals(event, "CLICKED")) {
      set(img, charAt(name,1)-'0', charAt(name,3)-'0', randomColor());
    }
  }
}
```



RAPPEL IMPORTANT

- **Interdit de travailler en mode graphique tant que vous n'avez pas la validation d'une version en mode texte totalement fonctionnelle de votre projet !**
- Le projet vise à exercer vos compétences en génie logiciel et pas en graphisme !
- Il y a déjà eu des projets en mode texte obtenant 20 et des projets graphiques en dessous de 10 ….