Cours 2 : Qu'est-ce que la programmation procédurale en Java?

Yann Régis-Gianas
yrg@pps.univ-paris-diderot.fr

Université Paris Diderot – Paris 7

1. Sortez un appareil qui peut se rendre sur l'internet (smartphone, ordinateur, tablette, ...) ;
2. **Si** il n'est pas connecté à internet, **alors** connectez-le ;
3. Ouvrez un navigateur à l'adresse suivante :

   `http://www.ip101.fr/quiz/2`

4. **Tant que** le prof ne parle pas, ne faites rien.
5. Ecoutez.

Quiz :

Comment évaluez-vous votre compréhension du cours précédent ?

Quiz :

Qu'est-ce qui vous a posé problème dans le premier défi ?

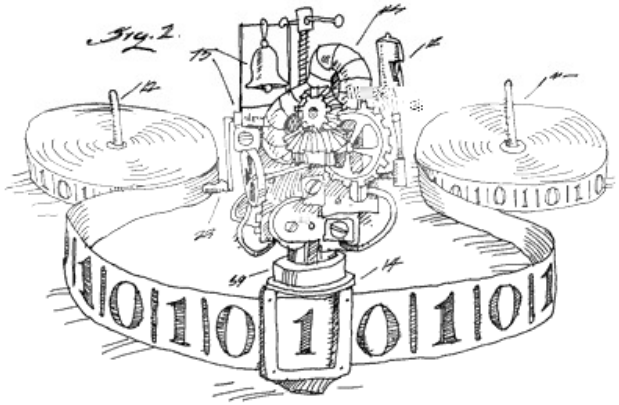
La résolution du défi

Le cours d'aujourd'hui

1. Striptease d'un ordinateur
2. La programmation procédurale en Java (en programmant un Pong)

L'ordinateur, c'est d'abord une idée

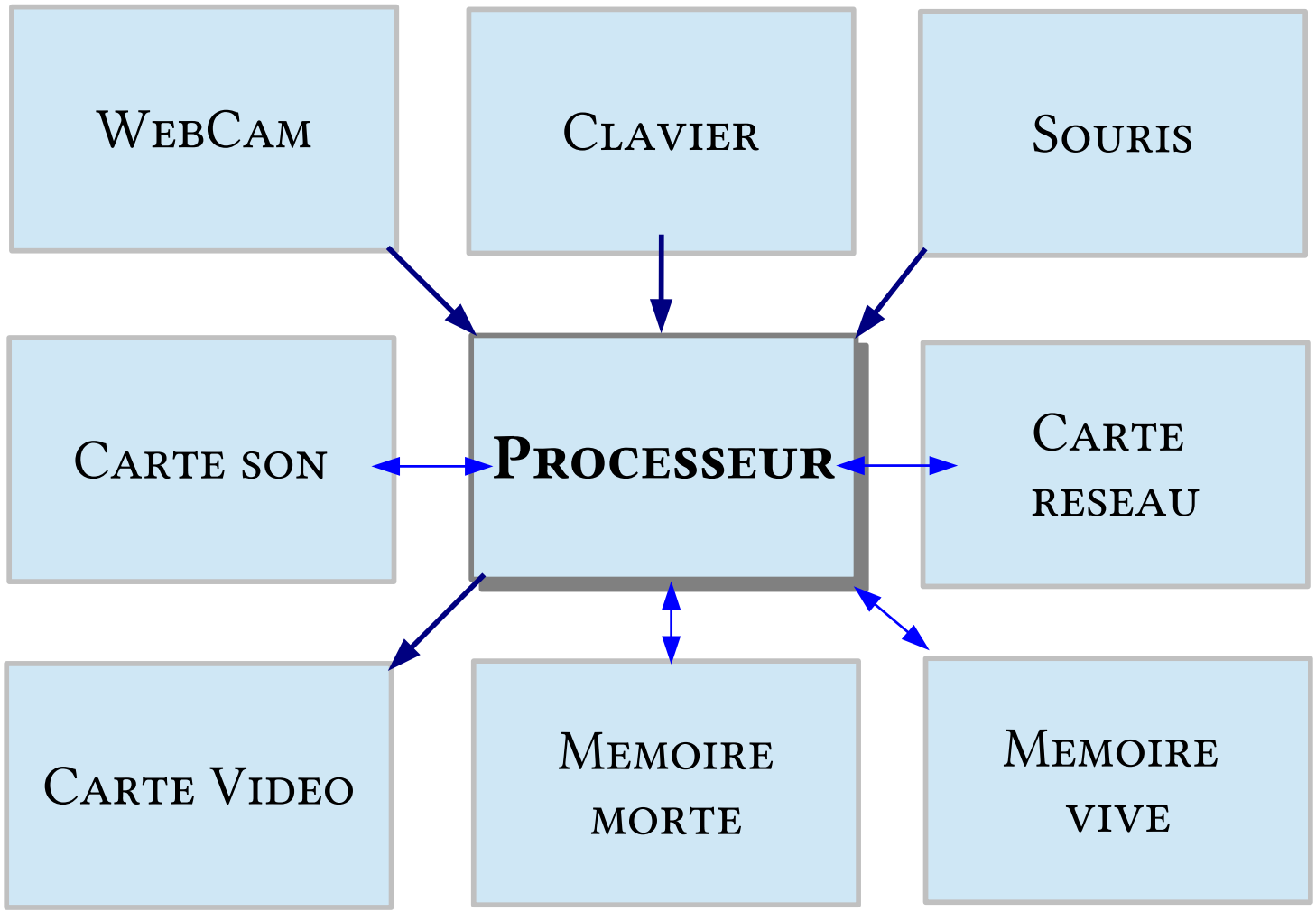
- Une mémoire
- Une unité de contrôle



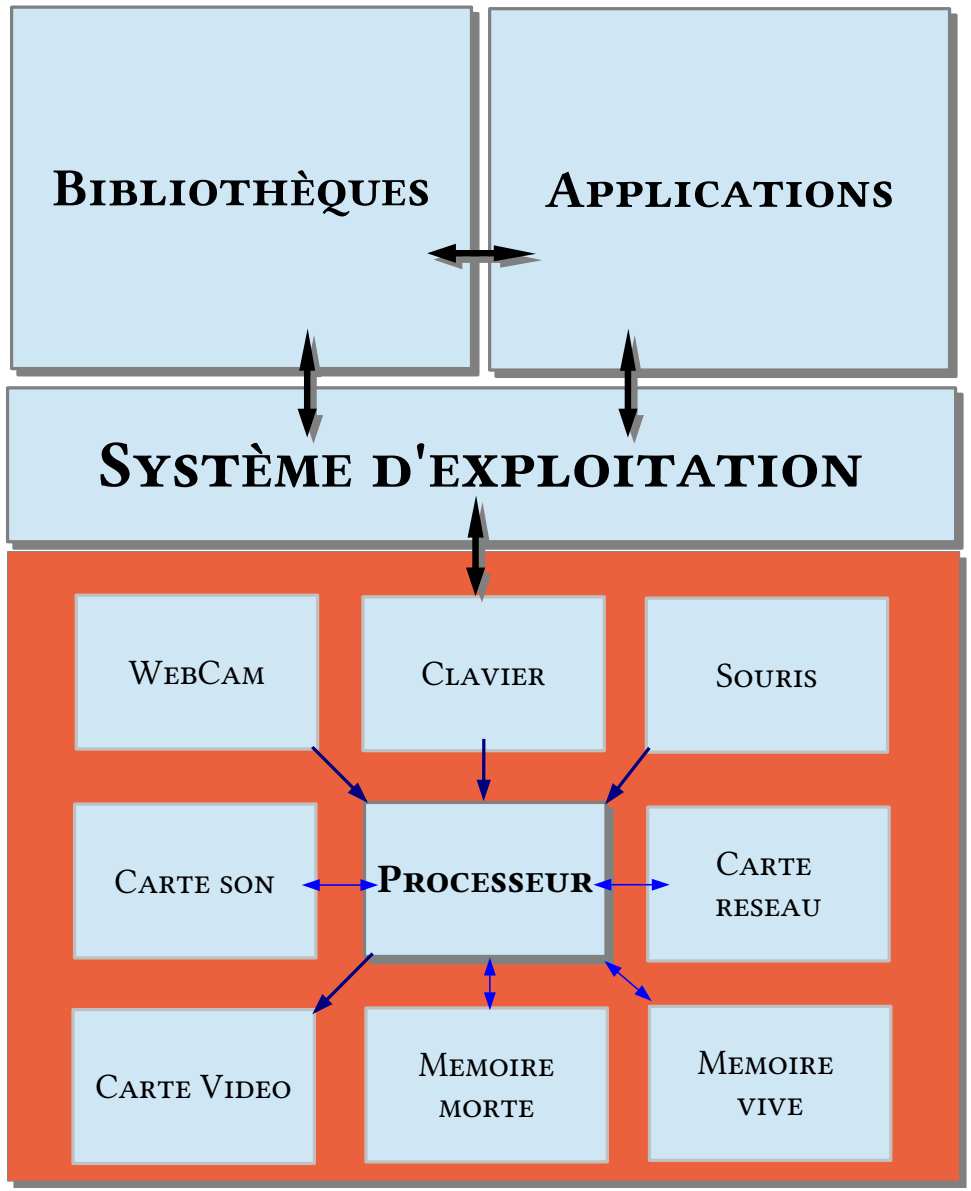
« Tout ce qui est calculable l'est par une machine de Turing. »

(La thèse de Church)

Un ordinateur, du point de vue matériel



Un ordinateur, du point de vue logiciel



Un peu de vocabulaire

Bit

Un **bit** est un chi□re en base binaire, valant donc 0 ou 1. C'est l'unité de mesure des quantités de données. Un **octet** est une séquence de 8 bits.

Système d'exploitation

Un **système d'exploitation** est le premier logiciel qu'exécute un ordinateur. Il sert à fournir des primitives d'accès aux fonctionnalités de l'ordinateur.

Fichier

Un **fichier** est une séquence d'octets stockée généralement en mémoire morte. Un fichier peut être créé, ouvert, lu, écrit et fermé. Le système d'exploitation s'occupe d'allouer un espace en mémoire morte à chaque fichier. Il structure aussi les fichiers sous la forme de **système de fichiers**.

Dans un système de fichiers, les fichiers sont généralement regroupés en **dossiers**. Les dossiers pouvant contenir eux-mêmes des dossiers. On obtient une **arborescence** de fichiers où chaque fichier est localisé par son **emplacement** et son **nom**.

Exemple d'arborescence de fichiers

```
|---lib64
|---.config
|---lost+found
|---bin
|---proc
|---tmp
|---lib32
|---sbin
|---home
|---lib
|---mnt
|---root
|---dev
|---etc
|---srv
|---usr
|---opt
|---.pulse
|---var
|---run
|---selinux
|---media
|---sys
|---boot
```

Exemple d'arborescence de fichiers

```
|---lib64
|---.config
|---lost+found
|---bin
|---proc
|---tmp
|---lib32
|---sbin
|---home
|---lib
|---mnt
|---root
|---dev
|---etc
|---srv
|---usr
|    |---bin
|    |---lib32
|    |---local
|    |---sbin
|    |---include
|    |---lib
|    |---share
|    |---games
|    |---src
|---opt
|---.pulse
|---var
|---run
|---selinux
|---media
```

Encore un peu de vocabulaire

Fichier exécutable

Certains fichiers sont des représentations de programmes qui peuvent être interprétées par le processeur. On les appelle des **fichiers exécutables** ou **programmes**.

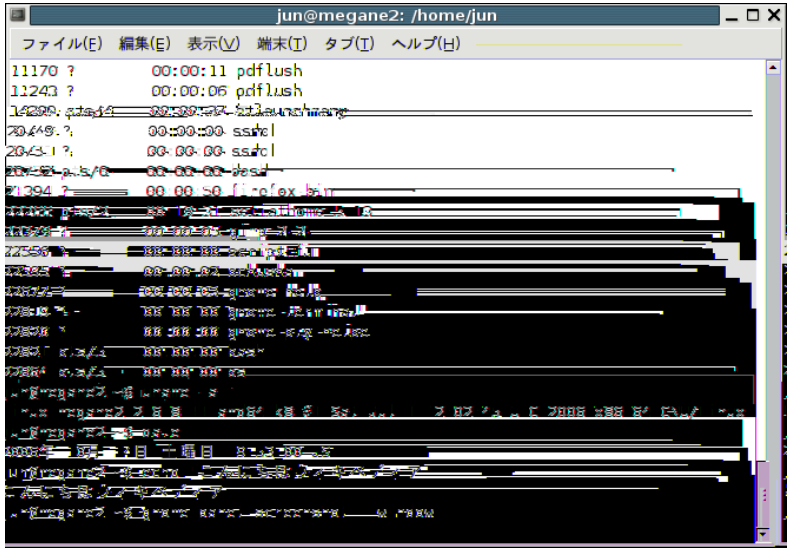
Bibliothèques

La plupart des programmes s'appuient sur des fonctionnalités similaires (ouvrir une fenêtre, envoyer des données sur le réseau, etc). Ces fonctionnalités sont souvent regroupées dans des **bibliothèques**, des fichiers dans lesquels les programmes exécutables peuvent aller puiser des primitives.

Interface utilisateur

Beaucoup de programmes interagissent avec des humains. Ils fournissent alors des moyens pour capturer les entrées produites par l'utilisateur (en cliquant, en écrivant des commandes, ...). Ils produisent aussi des visualisations sur l'écran ou sur les périphériques qu'ils ont à disposition. On dit que les programmes qui ont de telles fonctionnalités possèdent une **interface utilisateur**.

Exemple d'interfaces utilisateurs



L'exécution d'un programme

Exécution d'un programme

Lorsque l'utilisateur demande au système d'exploitation d'exécuter un programme, le code machine contenu dans le fichier du programme est chargé en mémoire et le processeur interprète les instructions qu'il contient.

Arguments d'un programme

Souvent les programmes prennent des **arguments**, c'est-à-dire des valeurs qui indique au programme exécuté comment on veut l'utiliser.

Exemple : la visualisation d'un document PDF

Dans un terminal UNIX, la commande suivante :

```
# evince slides.pdf
```

signifie :

« Lance le programme `evince` en lui donnant en paramètre la valeur `slides.pdf` »

(Ce paramètre est le nom du fichier du document à visualiser.)

La compilation d'un programme

Compilation

Compiler un programme, c'est exécuter le compilateur en lui indiquant quels fichiers contenant du code source doivent être traduits en programmes exécutables.

Et mon programme Java dans tout ça ?

Compilation en Java

Pour compiler un programme Java écrit dans un fichier nommé `Hello.java`, il su□t d'écrire :

```
# javac Hello.java
```

(`javac` est le nom du fichier du compilateur Java, `Hello.java` est le nom du fichier source que l'on veut compiler.)

Il peut alors se passer deux choses :

1. Votre code source n'a pas de sens car vous n'avez pas respecté la syntaxe de Java par exemple. Vous obtenez alors une **erreur de compilation**.
2. Votre code source a du sens. Le compilateur Java peut le traduire en un programme exécutable. Dans notre exemple, le programme exécutable s'appelera `Hello.class` car le fichier source s'appelle `Hello.java`.

Comment exécuter le programme Java obtenu ?

Le programme `java`

Les programmes Java sont exécutés à l'aide de la commande suivante :

```
# java NomDuProgramme
```

Par exemple, pour exécuter le programme du transparent précédent, on écrit :

```
# java Hello
```

Il existe d'autres erreurs...

Il existe d'autres erreurs...

Erreur d'exécution

Le programme suivant produit une **erreur à l'exécution** car la division par zéro est indéfinie.

```
x = 2 / 0;
```

Erreur dans l'algorithme

Un programme peut produire une valeur incorrecte. Par exemple :

```
// Le programme suivant ne stocke pas 2 puissance 1000
// dans la variable a.
int a = 2;
for (int i = 0; i < 1000; i++) {
   a = a * a;
}
```

Quiz :

Cette erreur est-elle une erreur à la compilation ou à l'exécution ?

La programmation procédurale

Programmation procédurale

La programmation procédurale construit des programmes à l'aide des mécanismes suivants :

- L'évaluation d'expressions (arithmétiques, booléennes, etc.)
- L'écriture et la lecture de variables.
- Les appels de procédures ou de fonctions.
- Le séquencement.
- Le branchement conditionnel.
- L'itération à l'aide de boucles (bornées ou non).
- La définition de procédures ou de fonctions.

La programmation procédurale en Java

Aujourd'hui, nous allons préciser la syntaxe *en Java* des différentes constructions de la programmation procédurale. Pour illustrer ces constructions, nous allons programmer le jeu PONG.

(Qu'est-ce que PONG ? La réponse ici : https://www.youtube.com/watch?v=iPigmdzAbyk)

Déclarations, instructions et expressions en Java

Classes syntaxiques de Java

Il y a trois types de constructions syntaxiques dans Java :

- Les **déclarations** :
  elles servent à associer des noms, aussi appelés **identifiants** à des définitions. Nous allons voir les déclarations de variables, de procédures et de fonctions.
- Les **instructions** :
  elles servent à e□ectuer des actions. Elles ne produisent pas de valeur mais un e□et.
- Les **expressions** :
  elles servent à calculer des valeurs. On les utilise par exemple pour décrire l'initialisation d'une variable, pour donner les arguments d'un appel de procédure ou encore pour représenter la condition d'un branchement conditionnel.

La sagesse de votre grand-mère s'applique à la syntaxe

« Une place pour chaque chose et chaque chose à sa place ! »

La sagesse de votre grand-mère s'applique à la syntaxe

« Une place pour chaque chose et chaque chose à sa place ! »

Comment maîtriser la syntaxe ?

1. Comprendre où écrire les déclarations, les instructions et les expressions.
2. Intégrer les règles de la syntaxe de chacune des constructions syntaxiques.

Les expressions arithmétiques

```
42
6 * 7
(3 + 4) * 12 / 2 - 12
```

Qu'est-ce que ça signifie ?

Une expression arithmétique e□ectue un calcul sur des entiers à l'aide des instructions arithmétiques de la machine. Toute expression a un **type**. Il représente l'ensemble des valeurs que peut prendre l'expression. Le type d'une expression arithmétique est **int**, c'est l'ensemble des valeurs entières comprises entre -2^{31} et $2^{31} - 1$.

Comment ça s'écrit ?

Une expression arithmétique peut être une constante entière ou bien le signe - suivi d'une expression arithmétique, ou bien une expression arithmétique suivie d'un opérateur binaire (comme $+$, $*$, $/$, $-$ ou %) suivi d'une expression arithmétique. Les priorités des opérateurs binaires suivent les conventions standards. On utilise des parenthèses pour choisir un ordre de calcul di□érent.

Les expressions arithmétiques

À quoi dois-je faire attention ?

- Il faut être sûr que l'on n'a pas oublié de parenthèses.

Les expressions booléennes

Qu'est-ce que ça signifie ?

Une expression booléenne fait des calculs logiques. Un calcul logique s'évalue en vrai (noté **true**) en Java ou en faux (noté **false** en Java). L'ensemble des valeurs $\{\textbf{true}, \textbf{false}\}$ est appelé type **boolean** en Java.

Comment ça s'écrit ?

Une expression booléenne peut être **true** ou bien **false** ou bien un opérateur de comparaison (==, !=, <, <=, >, >=) entre deux expressions arithmétiques, ou bien un opérateur logique binaire (||, &&) appliqué à deux expressions booléennes, ou bien l'opérateur de négation logique (!) appliqué à une expression booléenne.

Exemple :

```
x == 1
x ≥ 0 && x < 10
! (x ≥ 0 && x < 10)
```

À quoi dois-je faire attention ?

- À bien parenthéser pour calculer ce qu'il faut.

La déclaration des variables

```
int x = 0;
int ageOfTheCaptain = 2 * 20 + 2;
```

Qu'est-ce que ça signifie ?

La **déclaration d'une variable** réserve un espace en mémoire dans lequel sera stocké **une valeur du type de la variable**. Le type d'une variable représente l'ensemble des valeurs que peut prendre la variable. Par exemple, le type **int** représente l'ensemble des valeurs entières comprises entre -2^{31} et $2^{31} - 1$.

Comment ça s'écrit ?

On écrit d'abord le type de la variable, suivi de son nom, du signe égal et d'une expression dont la valeur sera la valeur initiale stockée dans la variable. Cette expression s'appelle l'**initialisation de la variable**.

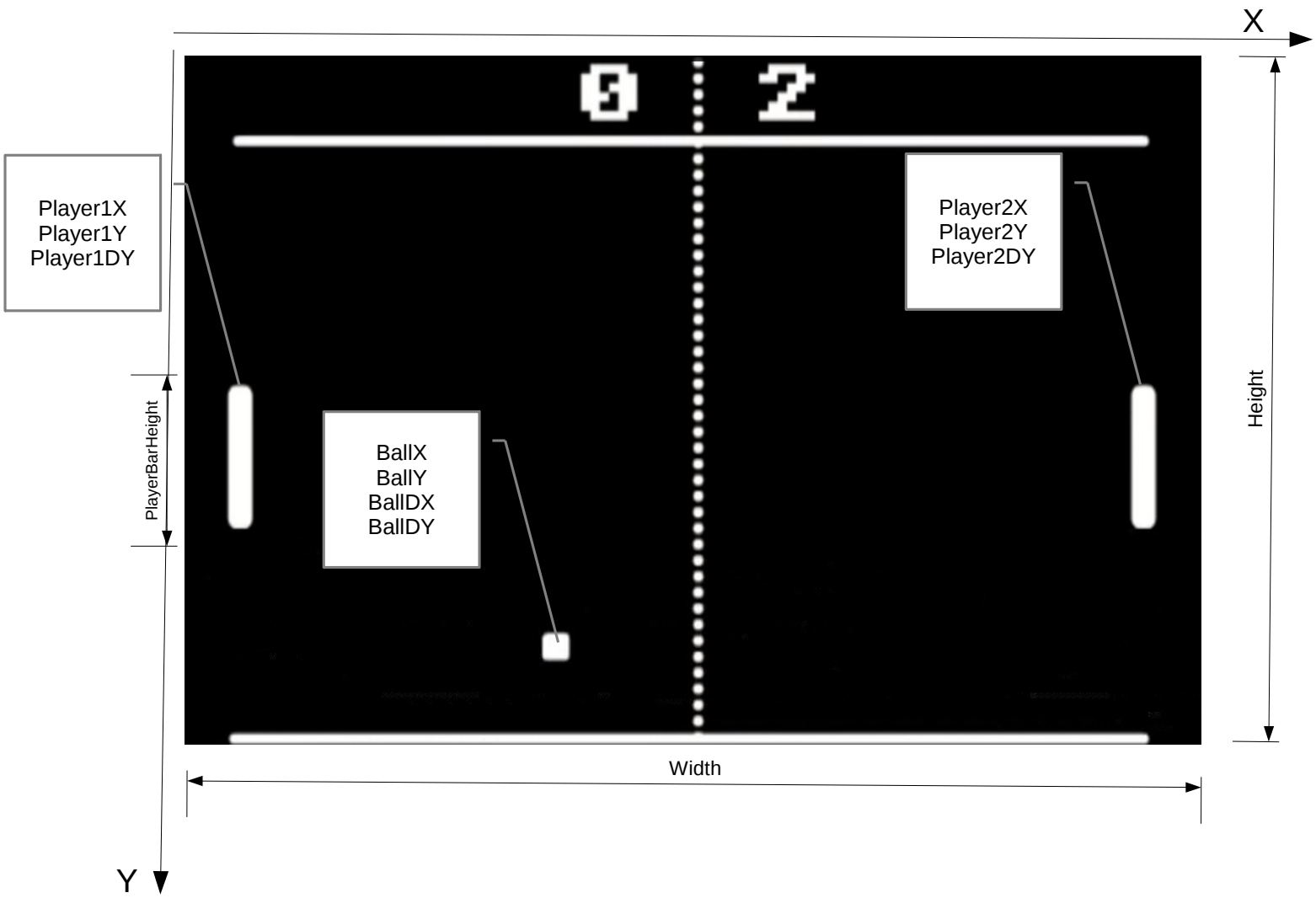
À quoi dois-je faire attention ?

- Il ne faut pas oublier le type, le nom et l'initialisation.

Quiz :

Parmi les fragments de code suivants, lesquels sont des initialisations valides de variables ?

Pong : Anatomie de l'état du jeu



Pong : Les variables représentant l'état du jeu

```
static int width = 1024;
static int height = 512;
static int blockSize = height / 30;
static int playerBarHeight = 7 * blockSize;
static int ballX = 0;
static int ballY = 0;
static int ballPrevX = 0;
static int ballPrevY = 0;
static int ballDX = 3;
static int ballDY = 3;
static int player1Score = 0;
static int player2Score = 0;
static int player1X = 0;
static int player1Y = 0;
static int player2X = width - blockSize;
static int player2Y = 0;
static int player1PrevY = 0;
static int player2PrevY = 0;
static int player1DY = 0;
static int player2DY = 0;
```

La variation des valeurs de ces variables représente l'évolution du jeu.

La lecture des variables

Qu'est-ce que ça signifie ?

Une expression qui lit une variable extrait la valeur de la variable au moment où on évalue l'expression,

Comment ça s'écrit ?

On écrit le nom de la variable dans l'expression.

Exemples :

```
int x = 6;
int ageOfTheCaptain = x * 7;
```

À quoi dois-je faire attention ?

- Il ne faut pas faire de faute d'orthographe dans l'identifiant de la variable.

L'appel d'une procédure

Qu'est-ce que ça signifie ?

Un appel de procédure est une instruction qui "donne la main" à une procédure puis continue l'exécution quand celle-ci à terminer son travail.

Comment ça s'écrit ?

Un appel de procédure est une instruction [1] qui commence par l'identifiant de la procédure suivi des arguments passés à cette procédure. Les arguments sont des expressions séparées par des virgules et entourées de parenthèses.
Exemple :

```
showTwoIntegers (42, 96) ;
```

À quoi dois-je faire attention ?

- Il ne faut pas de faute dans l'orthographe de l'identifiant.
- Il faut respecter le nombre et le type des paramètres de la procédure.

1. On ne peut donc pas faire d'appel de procédure dans une expression.

Quiz :

Parmi les instructions suivantes, quels sont les appels de procédures valides ?

Le séquencement

Qu'est-ce que ça signifie ?

Un séquencement est une instruction composée d'une liste d'instructions que l'on exécute dans l'ordre.

Comment ça s'écrit ?

Il su□t de faire suivre les instructions les unes à la suite des autres dans le code source.

Exemple :

```
showInteger (42) ;
showInteger (96) ;
```

À quoi dois-je faire attention ?

- Il ne faut pas oublier les points virgules.

Quiz :

Parmi les instructions suivantes, lesquels sont des séquencements valides ?

La déclaration d'une procédure

Qu'est-ce que ça signifie ?

Une déclaration d'une procédure nomme une séquence d'instructions – le **corps de la procédure** – à l'aide d'un identifiant.

Les valeurs de certaines variables utilisées dans le corps de la procédure sont laissées libres : ces variables sont les **paramètres** de la procédure. Ces valeurs sont fixées à l'aide de la valeur des arguments passés à chaque appel de la procédure.

La déclaration d'une procédure

Comment ça s'écrit ?

La déclaration d'une procédure commence par les mot-clés **static** et **void**, suivis de l'identifiant de la procédure et de la déclaration des paramètres. Les paramètres sont déclarés en écrivant le type suivi de l'identifiant du paramètre

Exemple :

```
static void ShowTwoIntegers (int x, int y) {
  ShowInteger (x) ;
  ShowInteger (y) ;
}
```

À quoi dois-je faire attention ?

- Attention à **static void**, aux parenthèses, aux accolades, aux virgules.
- Les paramètres doivent avoir des noms distincts.
- Ne pas oublier de déclarer tous les paramètres dont on a besoin.

Pong : Quelques procédures fournies

```
static void drawWhiteRectangle (int x, int y, int width, int height) ;
static void drawBlackRectangle (int x, int y, int width, int height) ;
```

Pong : Déclarer la procédure de jeu

```
static void player1 () {
}

static void player2 () {
}

static void ball () {
}

static void play () {
    player1 ();
    player2 ();
    ball ();
}
```

Pong : Déclarer la procédure de dessin

```
static void showPlayerBar (int xpos, int ypos) {
  drawWhiteRectangle (xpos, ypos, blockSize, playerBarHeight) ;
}

static void draw_player1 () {
  showPlayerBar (player1X, player1Y) ;
}

static void draw_player2 () {
  showPlayerBar (player2X, player2Y) ;
}

static void draw_ball () {
  drawWhiteRectangle (ballX, ballY, blockSize, blockSize) ;
}

static void draw () {
  draw_player1 () ;
  draw_player2 () ;
  draw_ball () ;
}
```

L'affectation des variables

Qu'est-ce que ça signifie ?

L'a□ectation d'une variable est une instruction qui change la valeur stockée dans une variable en la valeur d'une expression donnée.

Comment ça s'écrit ?

On écrit d'abord le nom de la variable suivi de l'expression donnant sa nouvelle valeur.

Exemple :

```
x = 1;
ageOfTheCaptain = 6 * 7;
```

À quoi dois-je faire attention ?

- À l'orthographe de l'identifiant de la variable.

Pong : Faire bouger la balle

```
static void ball () {
   ballX = ballX + ballDX;
   ballY = ballY + ballDY;
}
```

Quiz :

Que va-t-il se passer quand on exécute le programme ?

Pong : Faire bouger la balle (vraiment)

```
static void ball () {
   ballPrevX = ballX;
   ballPrevY = ballY;
   ballX = ballX + ballDX;
   ballY = ballY + ballDY;
}

static void draw_ball () {
   drawBlackRectangle (ballPrevX, ballPrevY, blockSize, blockSize);
   drawWhiteRectangle (ballX, ballY, blockSize, blockSize);
}
```

Les instructions conditionnelles

```
if (ageOfCaptain == 1) {
   ageOfCaptain = 6 * 7;
} else {
   ageOfCaptain = 0;
}
```

```
if (ageOfCaptain == 1) {
   ageOfCaptain = 6 * 7;
}
```

Qu'est-ce que ça signifie ?

Une **instruction conditionnelle** choisit d'exécuter une instruction plutôt qu'une autre en fonction de la valeur d'une expression booléenne, appelée **condition de la boucle**.

Plus précisément, la première instruction – aussi appelée première branche – est exécutée si la condition vaut **true** tandis que la seconde instruction est exécutée si la condition vaut **false**.

Les instructions conditionnelles

Comment ça s'écrit ?

On écrit le mot-clé **if**, puis une expression booléenne entre parenthèses, puis des instructions entre accolades, le mot-clé **else**, puis une autre série d'instructions entre accolades.

Quand il n'y a rien à faire si la condition est fausse, on peut ne pas écrire le mot-clé **else** suivi de sa série d'instructions entre accolades.

Quand une branche n'est fait que d'une instruction, on peut se passer des accolades.

Une composition fréquente d'instructions conditionnelles

Chaînage des instructions conditionnelles

De nombreux programmes consistent en **une série d'instructions conditionnelles**. Dans ces cas-là, on souhaite chaîner les instructions ce que l'on obtient en écrivant une instruction conditionnelle dans la seconde branche :

```
static boolean doesTheCaptainExists = true;
if (ageOfTheCaptain == 0) {
    doesTheCaptainExists = false;
} else if (ageOfTheCaptain ≥ 150) {
    doesTheCaptainExists = false;
} else if (ageOfTheCaptain ≥ 120 && theCaptainIsRunningMarathon) {
    doesTheCaptainExists = false;
}
```

Les instructions conditionnelles

À quoi dois-je faire attention ?

- Ne pas oublier le **if**, le **else**, les parenthèses et les accolades.
- À écrire une expression booléenne pour la condition.

Pong : Faire rebondir la balle

```
static void ball () {
 if (ballX < 0 || ballX > width) {
     ballX = width / 2;
     ballY = height / 2;
 } else if (ballY > height - blocksize || ballY < 0) {
     ballDY = -ballDY;
 } else if ((ballX ≤ blockSize)
              && (ballY ≥ player1Y && ballY ≤ player1Y + playerBarHeight))
{
     ballDX = -ballDX;
 } else if ((ballX ≥ width - blockSize)
              && (ballY ≥ player2Y && ballY ≤ player2Y + playerBarHeight)) {
    ballDX = -ballDX;
 }
}
```

Pong : Faire rebondir la balle

```
static void ball () {
 if (ballX < 0 || ballX > width) {
     ballX = width / 2;
     ballY = height / 2;
 } else if (ballY > height - blocksize || ballY < 0) {
     ballDY = -ballDY;
 } else if ((ballX ≤ blockSize)
              && (ballY ≥ player1Y && ballY ≤ player1Y + playerBarHeight))
{
     ballDX = -ballDX;
 } else if ((ballX ≥ width - blockSize)
              && (ballY ≥ player2Y && ballY ≤ player2Y + playerBarHeight)) {
    ballDX = -ballDX;
 }
}
```

Ce fragment de code n'est pas très lisible...
Nous allons grandement améliorer sa lisibilité dans quelques transparents.

Les appels de fonctions

Qu'est-ce que ça signifie ?

Un **appel de fonction** est une expression dont l'évaluation "donne la main" à une fonction et s'évalue en la valeur retournée par cette fonction.

Comment ça s'écrit ?

Comme un appel de procédure, sauf que l'on fait des appels de fonction dans des expressions (car un appel de fonction n'est pas une instruction).

Exemple :

```
exponent (2, 10)
4 * exponent (2, 10)
```

À quoi dois-je faire attention ?

- À ne pas oublier d'arguments et à fournir des arguments du bon type.

La déclaration d'une fonction

Comment ça s'écrit ?

La déclaration d'une fonction commence par le mot-clé **static** suivi du type de la valeur retournée par la fonction, suivis de l'identifiant de la fonction et de la déclaration des paramètres. Les paramètres sont déclarés en écrivant le type suivi de l'identifiant du paramètre comme pour les procédures.

Pour indiquer la valeur à renvoyer par la fonction, on utilise le mot-clé **return** suivi de l'expression qui calcule cette valeur à retourner.

Exemple :

```
static int MeanOfTwoIntegers (int x, int y) {
  return ((x + y) /2);
}
```

À quoi dois-je faire attention ?

- Attention à **static**, aux parenthèses, aux accolades, aux virgules.
- Il faut que le type du retour soit le même que celui de la valeur retournée.
- Les paramètres doivent avoir des noms distincts.
- Ne pas oublier de déclarer tous les paramètres dont on a besoin.

Pong : Les fonctions fournies

```
static boolean isUpPressed ();
static boolean isDownPressed ();
static boolean isAPressed ();
static boolean isQPressed ();
```

Pong : Faire bouger les joueurs

```
static void player1 () {
   player1PrevY = player1Y;
   if (isUpPressed ()) {
       player1DY = -5;
   } else if (isDownPressed ()) {
      player1DY = 5;
   } else {
     player1DY = 0;
   }
   player1Y = player1Y + player1DY;
}
```

(Même chose pour `player2`, mais en changeant les touches.)

Pong : Retour sur la balle

```
static boolean ballIsOut () {
 return (ballX < 0 || ballX > width)
}

static boolean ballCollidesFloorOrCell () {
 return (ballY > height - blocksize || ballY < 0) ;
}

static boolean ballCollidesPlayer1 () {
   return (ballX + ballDX ≤ blockSize
            && (ballY ≥ player1Y && ballY ≤ player1Y + playerBarHeight) ;
}

static boolean ballCollidesPlayer2 () {
   return (ballX + ballDX ≥ width - blockSize
            && (ballY ≥ player2Y && ballY ≤ player2Y + playerBarHeight) ;
}
```

Pong : Retour sur la balle

```
static void ball () {
 if (ballIsOut ()) {
      ballX = width / 2;
      ballY = height / 2;
 } else if (ballCollidesFloorOrCell ()) {
      ballDY = -ballDY;
 } else if (ballCollidesPlayer1 ()) {
    ballDX = -ballDX;
 } else if (ballCollidesPlayer2 ()) {
    ballDX = -ballDX;
 }
}
```

Pong : Retour sur la balle

```
static void ball () {
 if (ballIsOut ()) {
     ballX = width / 2;
     ballY = height / 2;
 } else if (ballCollidesFloorOrCell ()) {
     ballDY = -ballDY;
 } else if (ballCollidesPlayer1 () || ballCollidesPlayer2 ()) {
    ballDX = -ballDX;
 }
}
```

Pong : Retour sur la balle

```
static void ball () {
 if (ballIsOut ()) {
      ballX = width / 2;
      ballY = height / 2;
 } else if (ballCollidesFloorOrCell ()) {
      ballDY = -ballDY;
 } else if (ballCollidesPlayer1 () || ballCollidesPlayer2 ()) {
    ballDX = -ballDX;
 }
}
```

Avec le bon vocabulaire, les programmes sont plus clairs !

La boucle bornée

Qu'est-ce que ça signifie ?

Une **boucle bornée** répète zéro, une ou plusieurs fois une certaines séquences d'instructions.

Comment ça s'écrit ?

La boucle bornée est une instruction qui commence par le mot-clé **for** puis une parenthèse ouvrante, puis la déclaration de la **variable d'itération**, aussi appelée **compteur**, puis vient l'expression de **condition d'arrêt de la boucle**, puis **l'instruction d'incrémentation** du compteur, puis la parenthèse fermante et enfin les instructions à répéter entre accolades.

Exemple :

```
int sum = 0;
for (int i = 0; i < 5; i++) {
    sum = sum + i;
}
```

À quoi dois-je faire attention ?

- Ce sont des points-virgules qui séparent les composants d'une boucle.

Pong : Dessiner la délimitation des camps

```
static void drawDelimiters () {
   int dot_size = block_size / 2;
   int dotX = width / 2 - dot_size /4;
   int dotNb = height / dot size / 2;
   for (int y = 0; y < dotNb; y++) {
      drawWhiteRectangle (dotX, y * dot_size * 2, dot_size, dot_size);
   }
}
```

La boucle non bornée

Qu'est-ce que ça signifie ?

Une **boucle non bornée** répète zéro, une ou plusieurs fois une certaines séquences d'instructions **tant qu'une condition est vraie**.

Comment ça s'écrit ?

La boucle non bornée est une instruction qui commence par le mot-clé **while** suivi d'une expression booléenne entre parenthèses, suivie de la séquence d'instructions à répéter entre accolades.

Exemple :

```
int sum = 0;
int i = 0;
while (i < 5) {
  sum = sum + i;
  i = i + 1;
}
```

À quoi dois-je faire attention ?

- Une boucle non bornée peut ne jamais terminer !

Pong : Dessiner la délimitation des camps

```
static void drawDelimiters () {
   int dot_size = block_size / 2;
   int dotX = width / 2 - dot_size /4;
   int y = 0;
   while (y < height) {
      drawWhiteRectangle (dotX, y, dot_size, dot_size);
      y = y + 2 * dot_size;
   }
}
```

Pong : Dessiner la délimitation des camps

```
static void drawDelimiters () {
   int dot_size = block_size / 2;
   int dotX = width / 2 - dot_size /4;
   int y = 0;
   while (y < height) {
      drawWhiteRectangle (dotX, y, dot_size, dot_size);
      y = y + 2 * dot_size;
   }
}
```

Il est souvent compliqué, ou même impossible, de déterminer le nombre d'itérations à faire dans un programme. À l'aide d'une condition d'arrêt plus générale, il n'est pas nécessaire de déterminer ce nombre d'itérations.

Les expressions de type chaînes de caractères

Qu'est-ce que ça signifie ?

Une expression de type chaînes de caractères calcule du texte. Un texte est une séquence de caractères. Un caractère est la représentation (ici numérique) d'un symbole. En Java, le type des chaînes de caractères est **String**.

Comment ça s'écrit ?

Une expression de type **String** peut être un texte écrit entre guillemets doubles ou bien une expression de type **String** suivi d'un +, suivi d'une expression de type **String**. Cette dernière construction construit la **concaténation** de la seconde chaîne à la suite de la première chaîne.

Exemple :

```
String hello = "Bonjour";
String helloLuke = hello + "Luke";
```

Les expressions de type chaînes de caractères

À quoi dois-je faire attention ?

- À bien fermer les guillemets.
- À utiliser \" pour représenter un guillemet.
- À utiliser \n pour représenter un retour à la ligne.

Pong : Afficher le score

```
static void putString (int x, int y, String text) ;
static string intToString (int x) ;

static void showScore () {
   String message =
      intToString (player1Score) + " - " + intToString (player2Score) ;
   drawString (width / 2 - characterWidth, 0, message) ;
}
```

Ce qu'il nous reste à voir

Instructions

- Pas de nouvelles instructions mais de nouvelles façons de les **composer**.

Expressions

- Un nouveau type de données : les tableaux.
- À l'aide des tableaux, nous pourrons représenter de nombreuses données : des images, du son, des graphes, de l'ADN, etc.

Le deuxième défi

Que font ces programmes mystérieux ?