

Chapitre 2: Objets et Types de base

Présentation pour

Java Software Solutions

Foundations of Program Design

Deuxième Edition

par John Lewis et William Loftus

Java Software Solutions publié par by Addison-Wesley

Presentation slides are copyright 2000 by John Lewis and William Loftus. All rights reserved.

Instructors using the textbook may use and modify these slides for pedagogical purposes.

Objets et types de base

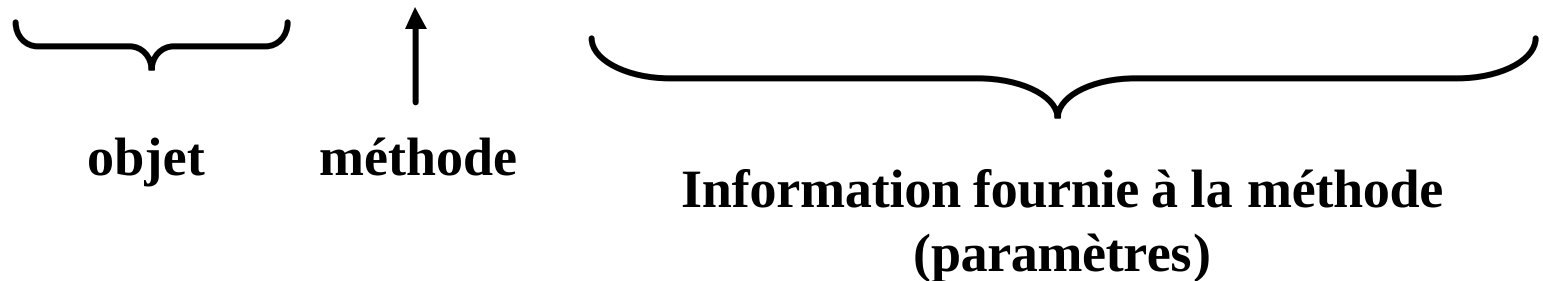


- **On peut maintenant explorer quelques concepts fondamentaux en programmation**
- **Chapitre 2 se concentre sur :**
 - **Objets prédéfinis**
 - **Types de données de données**
 - **La déclaration et l'utilisation de variables**
 - **Les expressions et la précedence des opérateurs**
 - **Les librairies de classes**
 - **Les applets Java**
 - **Le dessin de formes**

Introduction aux objets

- ⊗ A ce stade-ci, on peut penser qu'un *objet* est une collection de services qu'on peut lui demander d'effectuer
- ⊗ Les services sont définis par des méthodes dans une classe qui définit l'objet
- ⊗ Dans le programme `Lincoln`, nous appelons la méthode `println` de l'objet `System.out` :

```
System.out.println ("Whatever you are, be a good one.");
```



Les méthodes println et print

- ✿ L'objet `System.out` fournit d'autres services
- ✿ La méthode `print` est similaire à la méthode `println`, sauf qu'elle n'avance pas à la prochaine ligne
- ✿ Ainsi tout ce qui est imprimé après l'énoncé `print` apparaîtra sur la même ligne
- ✿ Voir `Countdown.java` (page 53)

Abstraction

- Une *abstraction* cache (ou ignore) les détails au bon moment
- Un objet est abstrait en ce qu'on n'a pas à penser à tous les détails internes pour pouvoir l'utiliser
- On n'a pas à savoir comment la méthode `println` fonctionne pour l'appeler
- Un humain peut seulement traiter sept (plus ou moins deux) informations à un moment donné
- Si on regroupe l'information en morceaux (tels les objets), on peut traiter plusieurs morceaux compliqués en même temps
- Ainsi, on peut écrire un software complexe en l'organisant en classes et en objets adéquats

La classe String

- Chaque chaîne de caractères est un objet en Java, défini par la classe **String**
- Chaque chaîne littérale, délimitée par des guillemets, représente un objet **String**
- *L'opérateur de concaténation de chaînes de caractères (+) est utilisé pour joindre une chaîne de caractères à la fin d'une autre*
- Il peut aussi être utilisé pour ajouter un nombre à une chaîne de caractères
- Une chaîne littérale ne peut pas être brisée sur deux lignes dans un programme
- Voir Facts.java (page 56)

Concaténation de chaînes de caractères

- L'opérateur plus (+) est aussi utilisé pour l'addition arithmétique
- La fonction effectuée par l'opérateur + dépend du type de l'information sur lequel il est appliqué
- Si les deux opérandes sont des chaînes de caractères, ou si un est une chaîne de caractères et l'autre est un nombre, il effectue une concaténation de chaînes de caractères
- Si les deux opérandes sont des nombres, il les additionne
- L'opérateur + est évalué de gauche à droite
- Des parenthèses peuvent être utilisées pour forcer un ordre dans les opérations
- Voir Addition.java (page 58)

Escape Sequences

- Comment imprimer le caractère guillemet (“)
- La ligne suivante rendrait le compilateur confus parce qu’il pourrait interpréter le deuxième guillemet comme la fin de la chaîne de caractères

```
System.out.println ("I said "Hello" to you.");
```

- Une *escape sequence* est une série de caractères qui représente un caractère spécial
- Une escape sequence commence par le caractère backslash (\), qui indique que le(s) caractère(s) qui suit doit être traité d’une façon spéciale

```
System.out.println ("I said \"Hello\" to you.");
```

Escape Sequences

• Quelques escape sequences en Java :

<u>Escape Sequence</u>	<u>Signification</u>
<code>\b</code>	backspace
<code>\t</code>	tab
<code>\n</code>	nouvelle ligne
<code>\r</code>	retour de chariot
<code>\"</code>	guillemet
<code>\'</code>	apostrophe
<code>\\</code>	backslash

• Voir Roses.java (page 59)

Variables

- Une *variable* est un nom pour un emplacement en mémoire
- Une variable doit être *déclarée*, spécifiant le nom et le type d'information qui sera contenu dans cet emplacement

type de donnée

nom de variable



```
int total;
```

```
int count, temp, result;
```

Plusieurs variables peuvent être créées dans une même déclaration

Variables

- On peut donner une valeur initiale à une variable dans sa déclaration

```
int sum = 0;  
int base = 32, max = 149;
```

- Lorsqu'une variable est référencée dans un programme, sa valeur courante est utilisée
- Voir PianoKeys.java (page 60)

Assignment

- **Un énoncé d'assignation change la valeur d'une variable**
- **L'opérateur d'assignation est le signe =**

`total = 55;`



- **L'expression de droite est évaluée et le résultat est stocké dans la variable à gauche**
- **La valeur qui était dans la variable `total` est écrasée**
- **On ne peut assigner une valeur à une variable que si elle est consistante avec le type déclaré de cette variable**
- **Voir Geometry.java (page 62)**

Constantes

- Une constante est un identificateur semblable à une variable, excepté qu'elle ne contient qu'une seule valeur durant toute son existence
- Le compilateur donnera une erreur si on essaie de modifier une constante
- En Java, le modificateur `final` déclare une constante

```
final int MIN_HEIGHT = 69;
```

- Constante :
 - Fournit un nom à des valeurs qui pourraient être ambiguë
 - Facilite les modifications au code
 - Réduit des erreurs par mégarde

Types de base

- Il y a exactement huit types de base pour les données en Java
- Quatre types représentent des entiers :
 - byte, short, int, long
- Deux types représentent des nombres en point flottant :
 - float, double
- Un type représente des caractères :
 - char
- Un type représente des valeurs booléennes :
 - boolean

Types numériques de base

- La différence entre les différents types numériques de base est leur taille, et ainsi les valeurs qu'ils peuvent stocker :

<u>Type</u>	<u>Stockage</u>	<u>Valeur Min</u>	<u>Valeur Max</u>
byte	8 bits	-128	127
short	16 bits	-32,768	32,767
int	32 bits	-2,147,483,648	2,147,483,647
long	64 bits	$< -9 \times 10^{18}$	$> 9 \times 10^{18}$
float	32 bits	+/- 3.4×10^{38} avec 7 chiffres significatifs	
double	64 bits	+/- 1.7×10^{308} avec 15 chiffres significatifs	

Caractères

- Une variable de type `char` stocke un seul caractère de *l'ensemble de caractères Unicode*
- Un *ensemble de caractères* est une liste ordonnée de caractères, et chaque caractère correspond à un nombre unique
- L'ensemble de caractères Unicode utilise 16 bits par caractère, permettant 65,536 caractères uniques
- C'est un ensemble de caractères international, contenant des symboles et des caractères de plusieurs langues du monde
- Les caractères littéraux sont délimités par un apostrophe :

`'a'`

`'X'`

`'7'`

`'$'`

`','`

`'\n'`

Caractères

- ***L'ensemble de caractères ASCII est plus ancien et plus petit que celui Unicode, mais il est encore très populaire***
- **Les caractères ASCII forment un sous-ensemble de l'ensemble de caractères Unicode, incluant :**

Lettres majuscules	A, B, C, ...
Lettres minuscules	a, b, c, ...
Ponctuation	point, point-virgule, ...
Chiffres	0, 1, 2, ...
Symboles spéciaux	&, , \, ...
Caractères de contrôle	retour de chariot, tab, ...

Booléen

- Une valeur **boolean** représente une condition vraie ou fausse
- Un booléen peut aussi être utilisé pour représenter les paires d'états, tel que l'ampoule est allumée ou éteinte
- Les mots réservés **true** et **false** sont les deux seules valeurs valides pour un type booléen

```
boolean done = false;
```

Expressions arithmétiques

- Une *expression* est une combinaison d'opérateurs et d'opérandes
- Les *expressions arithmétiques* calculent des résultats numériques et utilisent les opérateurs arithmétiques :

Addition	+
Soustraction	-
Multiplication	*
Division	/
Reste	%

- Si un seul ou les deux opérandes d'un opérateur arithmétique sont en point flottant, le résultat est en point flottant

Division et Reste

- Si les deux opérandes d'un opérateur division (/) sont des entiers, le résultat est un entier (la partie fractionnaire est éliminée)

14 / 3 égale? 4

8 / 12 égale? 0

- L'opérateur reste (%) retourne le reste après une division entière (premier opérande – (premier opérande / second opérande) * second opérande)

14 % 3 égale? 2

8 % 12 égale? 8

Précédence des opérateurs

- Les opérateurs peuvent être combinés en des expressions complexes

```
result = total + count / max - offset;
```

- Les opérateurs ont une précedence bien définie qui détermine l'ordre dans lequel ils sont évalués
- Multiplication, division, et reste sont évalués avant addition, soustraction, et concaténation de chaînes de caractères
- Les opérateurs arithmétiques avec la même précedence sont évalués de gauche à droite
- Les parenthèses peuvent être utilisées pour forcer un ordre d'évaluation

Précédence des opérateurs

❁ Quel est l'ordre d'évaluation des expressions suivantes ?

$$a + b + c + d + e$$

1 2 3 4

$$a + b * c - d / e$$

3 1 4 2

$$a / (b + c) - d \% e$$

2 1 4 3

$$a / (b * (c + (d - e)))$$

4 3 2 1

L'assignation

- L'opérateur d'assignation a une précedence plus basse que les opérateurs arithmétiques

En premier l'expression
à la droite du = est évaluée

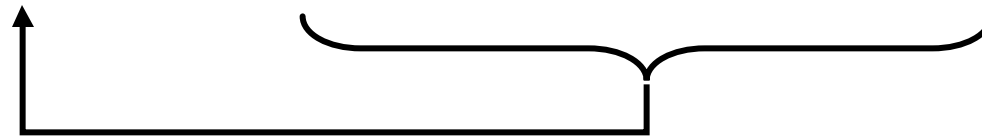
answer = sum / 4 + MAX * lowest;

4

1

3

2



Ensuite le résultat est stocké
dans la variable à gauche du =

L'assignation

- La droite et la gauche de l'assignation peuvent contenir la même variable

En premier, 1 est ajouté à la valeur originale de la variable count

`count = count + 1;`



Ensuite le résultat est stocké dans la variable count (effaçant la valeur originale)

Conversions de données

- ⊗ Il est parfois utile de convertir les données d'un type à un autre type
- ⊗ Par exemple, on peut vouloir traiter un entier en point flottant durant le calcul
- ⊗ Les conversions doivent être traitées avec soin pour ne pas perdre d'information
- ⊗ *Les conversions d'élargissement* transfèrent dans un type plus grand, elles sont plus sécuritaires (telles `short` vers `int`)
- ⊗ *Les conversions qui rappetissent le type* peuvent perdre de l'information (telles `int` vers `short`)

Conversions de données

- En Java, les conversions de données peuvent se produire de trois façons :
 - Conversion d'assignation
 - Promotion arithmétique
 - Casting
- La *conversion d'assignation* se produit quand une valeur d'un type est assignée à une variable d'un autre type
- Seulement des conversions d'élargissement peuvent se produire par assignation
- Une *promotion arithmétique* se produit automatiquement lorsque les opérateurs convertissent leurs opérandes

Conversions de données

- ❁ *Casting* est la plus puissante, et aussi la plus dangeureuse, technique pour convertir
- ❁ Des conversion d'élargissement et de rappetissement peuvent être accomplies avec un casting explicite d'une valeur
- ❁ Pour effectuer un cast, le type est inséré entre parenthèses devant la valeur à convertir
- ❁ Par exemple, si `total` et `count` sont des entiers, mais on veut un résultat en point flottant lors de leur division, on peut faire un cast devant `total`:

```
result = (float) total / count;
```

Créer des objets

- Une variable contient soit un type de base, ou elle contient une *référence* à un objet
- Un nom de classe peut être utilisé comme un type pour déclarer une *variable de référence à un objet*

`String title;`

- Aucun objet n'est créé lors de cette déclaration
- Une variable de référence à un objet contient l'*adresse* d'un objet
- L'objet lui-même doit être créé séparément

Créer des objets

- On utilise l'opérateur **new** pour créer un objet

```
title = new String ("Java Software Solutions");
```

Ceci appelle le *constructeur* de **String**, qui est une méthode spéciale qui initialise l'objet

- Créer un objet est appelé *instantiation*
- Un objet est une *instance* d'une classe particulière

Créer des objets

- Parce que les chaînes de caractères sont si communes, on n'a pas besoin d'utiliser l'opérateur `new` pour créer un objet `String`

```
title = "Java Software Solutions";
```

- Il s'agit d'une syntaxe spéciale qui ne fonctionne que pour les chaînes de caractères
- Une fois qu'un objet a été instantié, on peut utiliser l'opérateur `'.'` pour accéder à ses méthodes

```
title.length()
```

Méthodes de **String**

- La classe **String** a plusieurs méthodes qui sont utiles pour traiter des chaînes de caractères
- Plusieurs de ces méthodes *retournent une valeur*, telle un entier ou un nouvel objet de type **String**
- Voir la liste de méthodes de **String** à la page 75 et à l'appendice M
- Voir **StringMutation.java** (page 77)

Librairies de classes

- Une *librairie de classes* est une collection de classes qu'on peut utiliser en développant des programmes
- Il existe une *librairie standard de classes en Java* qui fait partie de tout environnement de développement en Java
- Ces classes ne font pas partie du langage Java à proprement parler, mais on se fie beaucoup sur elles
- La classe **System** et la classe **String** font partie de la librairie standard de classes en Java
- D'autres librairies de classes peuvent être acquises, ou on peut les créer soi-même

Packages

- Les classes de la librairie standard de classes en Java sont organisées en packages
- Quelques-uns des packages dans la librairie standard de classes en Java sont :

<u>Package</u>	<u>But</u>
<code>java.lang</code>	Support général
<code>java.applet</code>	Création d'applets pour le web
<code>java.awt</code>	Graphique et interfaces usager graphiques
<code>javax.swing</code>	Méthodes et composantes graphiques additionnelles
<code>java.net</code>	Communication réseau
<code>java.util</code>	Utilitaires

La déclaration import

- Quand on veut utiliser une classe d'un package, on peut utiliser son nom au complet

```
java.util.Random
```

- Ou on peut faire un *import* de la classe, et alors utiliser simplement le nom de la classe

```
import java.util.Random;
```

- Pour faire un *import* de toutes les classes dans un package particulier, on peut utiliser le caractère *

```
import java.util.*;
```

La déclaration import

- Toutes les classes du package `java.lang` sont automatiquement importées dans tous les programmes
- Ceci explique pourquoi on ne doit pas explicitement importer les classes `System` ou `String` dans les programmes précédents
- La classe `Random` fait partie du package `java.util`
- Elle fournit des méthodes qui génèrent des nombres pseudo-aléatoires
- On utilise souvent un *scale* et un *shift* d'un nombre pour le rendre dans un intervalle approprié
- Voir `RandomNumbers.java` (page 82)

Méthodes statiques

- Certaines méthodes peuvent être appelées avec le nom de la classe, plutôt qu'avec le nom d'un objet de la classe
- Ces méthodes sont appelées *méthodes de classe* ou *méthodes statiques*
- La classe `Math` contient plusieurs méthodes statiques, fournissant diverses fonctions mathématiques, telles une valeur absolue, les fonctions trigonométriques, la racine carrée, etc.

```
temp = Math.cos(90) + Math.sqrt(delta);
```

La classe Keyboard

- ✱ La classe **Keyboard** ne fait pas partie de la librairie standard de classes en Java
- ✱ Elle est fournie par les auteurs du livre pour rendre plus facile la lecture des entrées par le clavier
- ✱ Les détails de la classe **Keyboard** sont expliqués au Chapitre 8
- ✱ Pour l'instant, on ne fera que l'utiliser sans l'expliquer...
- ✱ La classe **Keyboard** fait partie du package appelé **cs1**, et elle contient plusieurs méthodes statiques pour lire des types spécifiques de données
- ✱ Voir Echo.java (page 86)
- ✱ Voir Quadratic.java (page 87)

Format de sortie

- La classe **NumberFormat** a une méthode statique qui retourne un objet pour formater

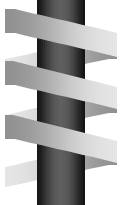
`getCurrencyInstance()`
`getPercentInstance()`

- Chaque objet pour formater a une méthode appelée **format** qui retourne une chaîne de caractères avec l'information spécifiée dans le format approprié
- Voir [Price.java](#) (page 89)

Format de sortie



- La classe **DecimalFormat** peut être utilisée pour formater une valeur en point flottant de façon générique
- Par exemple, on peut spécifier que le nombre soit imprimé avec trois positions après le point
- Le constructeur de la classe **DecimalFormat** a en paramètre une chaîne de caractères qui représente un format
- Voir CircleStats.java (page 91)



Applets

- Une application Java est un programme autonome avec une méthode `main` (comme celles vues jusqu'à présent)
- Un *applet* est un programme Java dont l'intention est d'être transporté sur le web et exécuté sur un web browser
- Un applet peut aussi être exécuté avec un outil `appletviewer` du Java Software Development Kit
- Un applet n'a pas de méthode `main`
- A la place, il a plusieurs méthodes spéciales qui répondent à des buts spécifiques
- La méthode `paint`, par exemple, est automatiquement exécutée et est utilisée pour dessiner le contenu d'un applet

Applets

- La méthode `paint` prend un paramètre qui est un objet de la classe `Graphics`
- Un objet `Graphics` définit un *contexte graphique* sur lequel on peut dessiner des formes et du texte
- La classe `Graphics` a plusieurs méthodes pour dessiner des formes
- La classe qui définit l'applet *étend* la classe `Applet`
- Elle utilise *l'héritage*, un concept orienté-objet couvert plus en détail au Chapitre 7
- Voir [Einstein.java](#) (page 93)

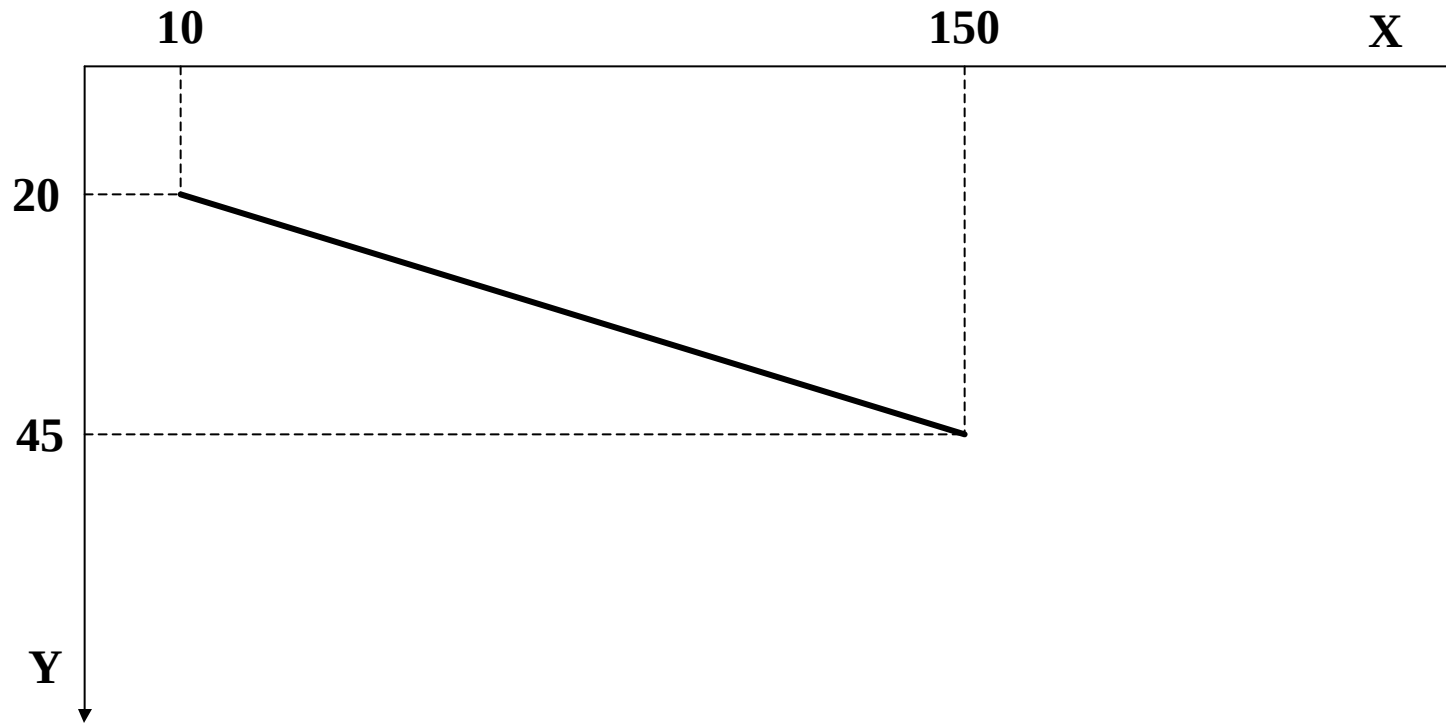
Applets

- ⊗ Un applet est inséré dans un fichier HTML en utilisant un *tag* qui fait référence au fichier bytecode de la classe applet
- ⊗ C'est la version bytecode du programme qui est transportée sur le web
- ⊗ L'applet est exécuté par un interpréteur Java qui fait partie du browser

Dessiner des formes

- **Regardons quelques méthodes de la classe `Graphics` qui dessinent des formes**
- **Une forme peut être remplie ou non, dépendant de la méthode appelée**
- **Les paramètres de la méthode spécifie les coordonnées et les tailles**
- **L'origine du système de coordonnées se trouve dans le coin supérieur gauche (Chapitre 1)**
- **Plusieurs formes courbes, comme un oval, sont dessinées en spécifiant son *rectangle englobant***
- **Un arc peut être pensé comme une section d'un oval**

Dessiner une ligne

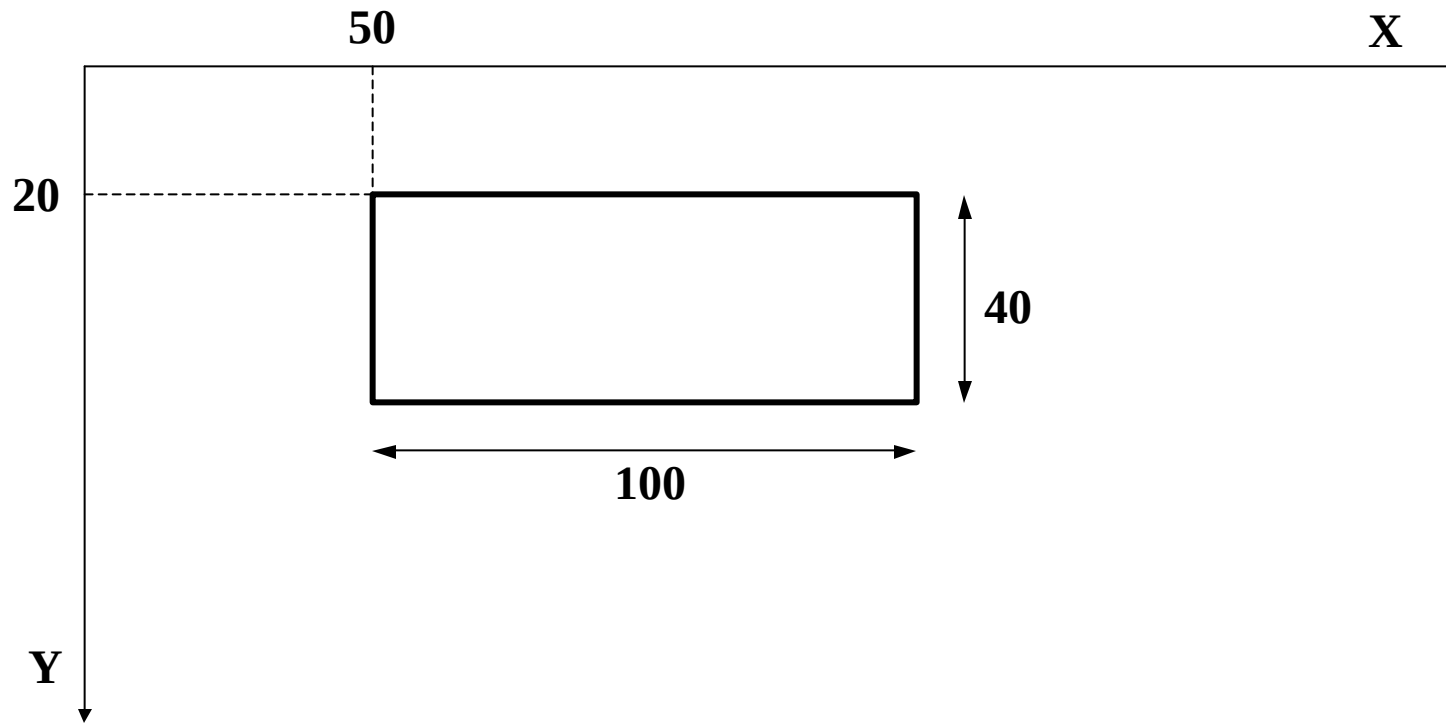


```
page.drawLine (10, 20, 150, 45);
```

ou

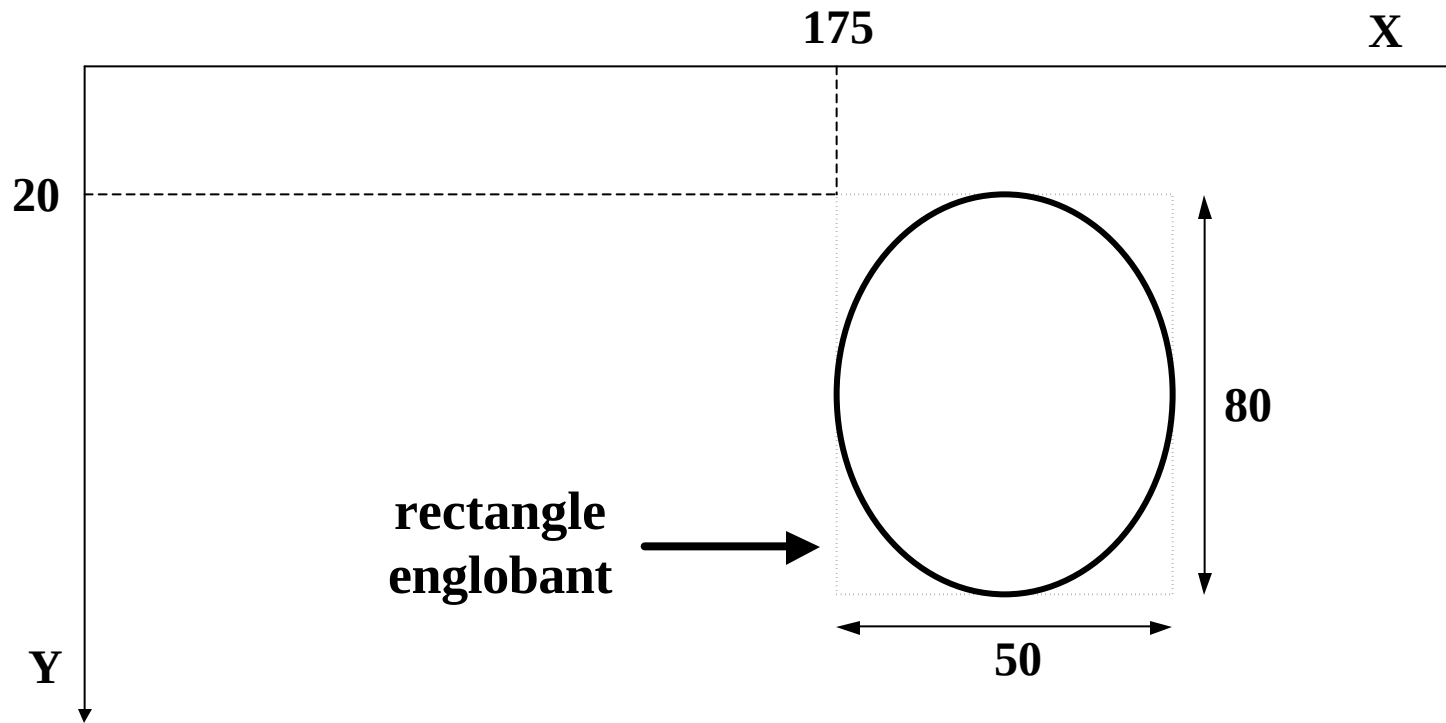
```
page.drawLine (150, 45, 10, 20);
```

Dessiner un rectangle



```
page.drawRect (50, 20, 100, 40);
```

Dessiner un oval



```
page.drawOval (175, 20, 50, 80);
```

La classe Color

- Une couleur est définie dans un programme Java en utilisant un objet créé par la classe **Color**
- La classe **Color** contient aussi plusieurs couleurs statiques prédéfinies
- Chaque contexte graphique a une couleur courante avant-plan (*foreground*)
- Chaque surface à dessiner a une couleur arrière-plan (*background*)
- Voir [Snowman.java](#) (page 99-100)