Algorithmique et programmation (1)

- Enoncé du problème
- Modèlisation du problème et de sa solution : Algorithmique
- Implémentation du problème et de sa solution sur machine : Programmation

Algorithmique et programmation (2)

- Deux approches pour définir l'algorithmique et sa frontière avec la programmation

  - Version 1 : L'algorithmique : c'est définir la logique procédurale en partant des données du problème pour aboutir à la solution

  - Version 2 : L'algorithmique : c'est définir les différents objets (avec leur mode de fonctionnement) qui permettent d'aboutir à la solution

Algorithmique et programmation (3)

- La version 1 est plus ancienne et elle a fait ses preuves. Elle atteint néanmoins ses limites pour au moins deux raisons qui sont liées :

  – Le langage d'abstraction est trop proche de la logique de la machine et donc pas assez de la logique de l'être humain

  – La modélisation du problème est très dépendante du langage de programmation

Algorithmique et programmation (4)

- Nous choisirons pour ce cours la version 2
  - Le principe essentiel pour l'algorithmique est de définir des objets en ne s'intéressant qu'aux fonctions de l'objet.
  - Le principe essentiel pour la programmation est de proposer pour chaque objet
    - une structure de données en mémoire
    - Le code qui reproduit fidèlement chaque fonction de l'objet

•Algorithmique et programmation (5)

- Le codage (programmation) des différentes fonctions doit respecter une règle essentielle
  - Les types des paramètres et le type de retour des fonctions codées doivent être strictement équivalentes à celles spécifiées dans la modélisation et être strictement indépendantes des structures de données en mémoire
- Cette règle garantit une évolution simple des programmes pour
  1. améliorer les performances
  2. définir de nouvelles fonctionnalités

Les types abstraits (1)

- On décrit les différents objets utilisés dans nos résolutions de problèmes à l'aide de types abstraits.

- Un type abstrait est un ensemble d'opérations décrivant ce que peuvent faire des objets même nature sans se soucier de la structure de l'objet.

Les types abstraits (2)

- Exemple : On peut décrire le fonctionnement d'une voiture à l'aide des opérations « démarrer », « tourner », « freiner », « accélérer » et « arrêter ». Ces opérations décrivent ce que sait faire n'importe quelle voiture

- On définit donc le type abstrait Voiture
  - démarrer
  - tourner
  - freiner
  - accélerer
  - arrêter

Les objets d'un type abstrait

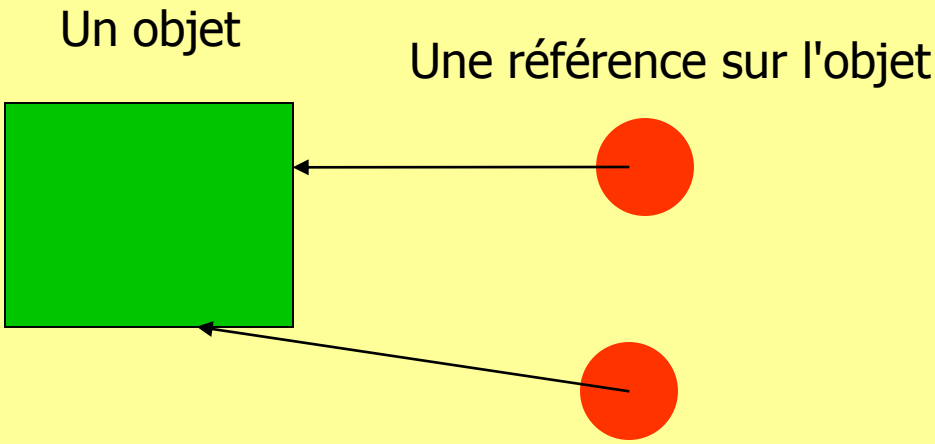
Les références

- Une référence permet de référer un objet
- Plusieurs références peuvent référer le même objet
- Une référence qui ne réfère rien vaudra **refNulle**

Une référence qui ne réfère sur aucun objet

La vie d'un objet

- Les objets ne seront manipulés qu'à l'aide de références
- Un objet doit d'abord être créé
- Une fois créé, il peut être transformé et observé à travers ses propriétés
- Un objet est automatiquement détruit quand il n'est plus référencé.

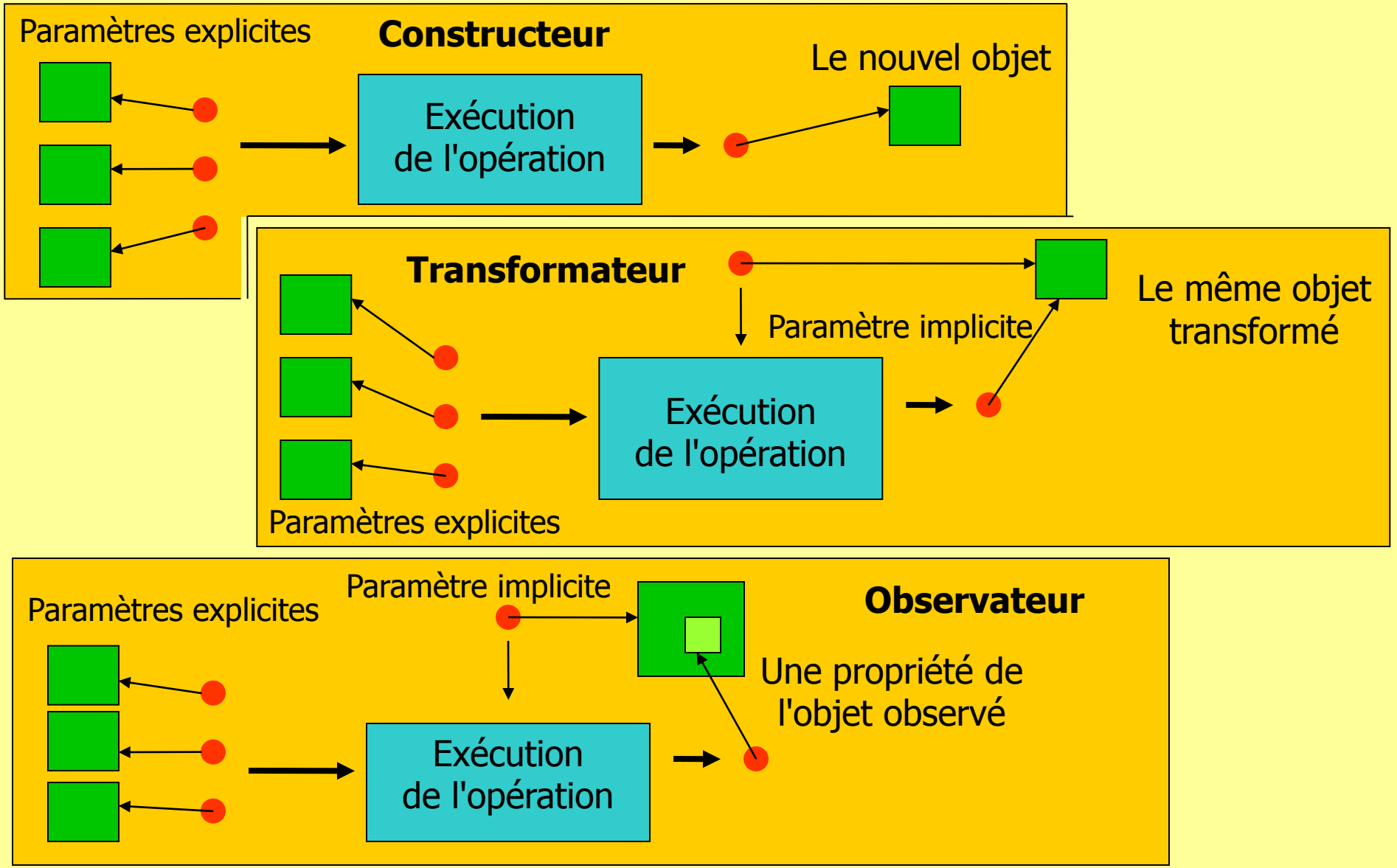


Les opérations (1)

- Les opérations d'un type abstrait permettent de décrire les traitements possibles que l'on peut faire avec des objets de ce type.

- On distingue trois genres d'opération
  - Les constructeurs
  - Les transformateurs
  - Les observateurs

Les opérations (2)



Définition d'une opération

- Pour définir une opération on doit préciser :
  - sa signature
  - ses pré-conditions
  - ce qu'elle fait à l'aide d'axiomes ou d'un algorithme

Signature d'une opération (1)

- La signature d'une opération permet de décrire la perception externe d'une opération
  1. Son genre
     - Constructeur : l'opération fabrique et renvoie une référence sur un nouvel objet
     - Transformateur : l'opération change l'état d'un objet existant et renvoie une référence de l'objet transformé
     - Observateur : l'opération renvoie une référence sur une propriété d'un objet existant
  2. Les types des paramètres explicites
     - C'est la liste des différents types d'objets que l'opération reçoit de l'extérieur lors d'un appel de cette opération

Signature d'une opération (2)

1. Son type de retour
   - C'est le type de l'objet que l'opération renverra à la fin de son exécution
     - Pour les constructeurs : ce type est celui du type abstrait auquel appartient l'opération
     - Pour les transformateurs : ce type est aussi celui du type abstrait auquel appartient l'opération
     - Pour les observateurs : le type de la propriété renvoyée

Pré-conditions d'une opération (1)

- C'est l'ensemble des conditions que doivent vérifier les paramètres implicites et explicites pour que l'opération puisse être exécutée.

- Si une opération est appelée en dehors de ces pré-conditions, l'exécution est interrompue et renvoie refNulle.

Pré-conditions d'une opération (2)

- Une pré condition utilise l'opération de tout type definie. Cette opération reçoit un paramètre x et retourne VRAI ou FAUX suivant que x réfère ou ne réfère pas un objet.

- Une pré-condition sur l'opération « op » est de la forme

definie(op(par1, par2, ...))

$\Leftrightarrow$

condition(par1, par2, ...)

Exemple n 1 d'une opération

- Soit l'opération qui permet de définir une équation du second degré
  - Les paramètres explicites sont des références de réels a, b et c coefficients de l'équation $a.x^2 + b.x + c = 0$
  - C'est un constructeur du type abstrait Eq2D donc pas de paramètre implicite.
  - Elle renvoie donc une référence d'un objet du type Eq2D
  - La pré condition est « a différent de 0 »

Exemple n 2 d'une opération

- Soit l'opération qui permet de multiplier le rayon d'un cercle
  - Le paramètre explicite est le coefficient multiplicateur « coeff »
  - C'est un transformateur du type abstrait Cercle
  - Elle renvoie donc une référence sur un objet du type Cercle
  - La pré condition est « coeff >= 0 ».

Exemple n 3 d'une opération

- Soit l'opération qui permet de récupérer le périmètre d'un cercle
  - Il n'y a pas de paramètre explicite
  - C'est un observateur du type abstrait Cercle
  - Elle renvoie donc une référence sur un objet du type abstrait Reel
  - Il n'y a pas de pré condition.

Description de l'action d'une opération

- Nous rappelons qu'un type abstrait décrit les opérations applicables à des objets mais ne décrit pas la structure en mémoire de ces objets (traité en programmation)

- On est donc amené à distinguer deux catégories d'opérations
  - Les opérations de base
  - Les opérations d'extension

Description d'une opération de base (1)

- Une opération de base est une opération qui dépend de la structure de l'objet
- Elle se décrit à travers des axiomes : expressions booléennes admises comme vraies par principe.
- Dans un premier temps, on l'exprimera en langage naturel sans utiliser de formalisme particulier

Description d'une opération de base (2)

- Exemple 1 : l'opération multiplierRayon du type abstrait Cercle :
  - ncerc <-- ocerc.multiplierRayon(coeff) $\Leftrightarrow$ le nouveau rayon du cercle référé par ncerc et ocerc est l'ancien rayon multiplié par coeff.
- Exemple 2 : l'opération creerEq2D du type abstrait EquationSecondDegre
  - eq $\leftarrow$ creerEq2D(a, b, c) $\Leftrightarrow$ eq réfère sur une équation de la forme « $a.x^2 + b.x + c = 0$ »

Description d'une opération d'extension (1)

- Une opération d'extension est une opération dont l'action peut être décrite de façon procédurale sans toucher à la structure des objets.
- On définira cette action à l'aide d'un algorithme
  - Sa signature
  - Déclaration du paramètre implicite s'il existe
  - Déclaration de références locales
  - L'ensembles des instructions à exécuter

Description d'une opération d'extension (2)

```
Observateur Eq2D calculerDelta() : Reel
Observé eq // paramètre implicite

Références locales
Reel a, b, c, delta

Debut
    a <- eq.recA()
    b <- eq.recB()
    c <- eq.recC()
    delta <- b * b – 4.0 * a * c
    retourner  delta
Fin
```

La partie déclarative de l'algorithme

Le corps de l'algorithme

Appel d'une opération (1)

- L'appel d'un construteur se fait comme suit :

  a <- monConstructeur(liste des paramètres explicites)

  où a est une référence du type abstrait de l'objet fabriqué.

- L'appel d'un observateur ou d'un transformateur se fait comme suit :

  a <- paramètre implicite.monOperation(liste des paramètres explicites)

  où a est une référence du type abstrait du retour de l'opération.

Appel d'une opération (2)

- L'appel provoque un arrêt momentané du déroulement de l'opération appelante pour exécuter l'opération appelée.
- La fin de l'exécution de l'opération appelée provoque un retour vers la fonction appelante (à l'endroit de l'appel) qui récupère le résultat de la fonction appelée.

Le type abstrait de base Booleen

- Type abstrait Booleen

Constructeur Booleen : vrai () : Booleen

Constructeur Booleen : faux () : Booleen

Constructeur Booleen : Booleen et Booleen : Booleen

Constructeur Booleen : Booleen ou Booleen : Booleen

Constructeur Booleen : non(Booleen b) : Booleen

Le type abstrait de base Entier

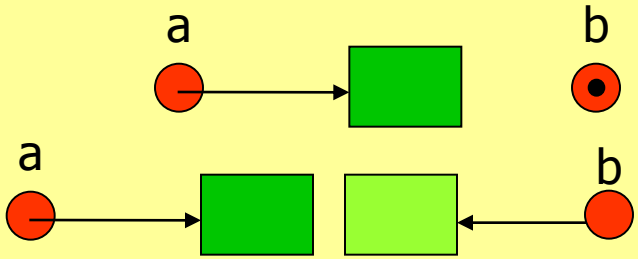
- Type abstrait Entier
  - Opérations de base
    - Constructeur Entier : zero () : Entier
    - Constructeur Entier : successeur (Entier) : Entier
    - Constructeur Entier : predecesseur (Entier) : Entier
  - Opérations d'extension
    - Constructeur Entier : Entier + Entier : Entier
    - Constructeur Entier : Entier - Entier : Entier
    - Constructeur Entier : Entier * Entier : Entier
    - Constructeur Entier : Entier div Entier : Entier
    - Constructeur Entier : Entier mod Entier : Entier
    - Observateur Entier : <= Entier : Booleen

Le type abstrait de base Reel

- Type abstrait Reel
  - Constructeur Reel : zero() : Reel
  - Constructeur Reel : Reel + Reel : Reel
  - Constructeur Reel : Reel - Reel : Reel
  - Constructeur Reel : Reel * Reel : Reel
  - Constructeur Reel : Reel / Reel : Reel
  - Constructeur Reel : racineCarree(Reel) : Reel
  - Observateur Reel : <= Reel : Booleen
- Pour utiliser un entier comme un réel, on définit l'opération suivante :
  - Constructeur Reel : conversion(Entier) : Reel

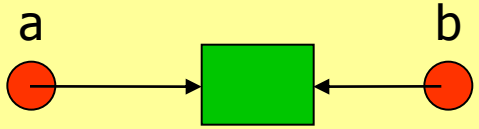
Les opérateurs et opérations globales (1)

- <- : l'opérateur d'affectation de référence



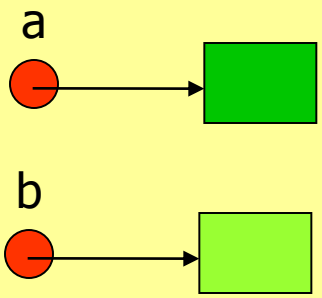
avant l'exécution

b <- a



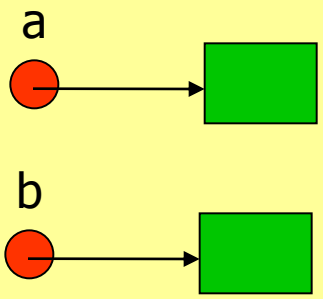
après l'exécution

- egalA : l'opération d'affectation de contenu



avant l'exécution

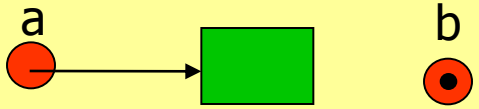
b.egalA(a)



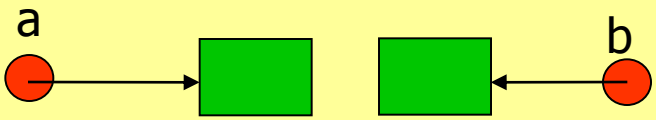
après l'exécution

Les opérateurs et opérations globales (2)

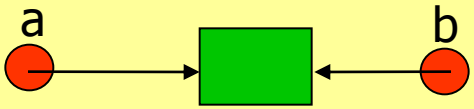
- = : l'opérateur d'égalité de référence



l'exécution de « a = b » renverra FAUX

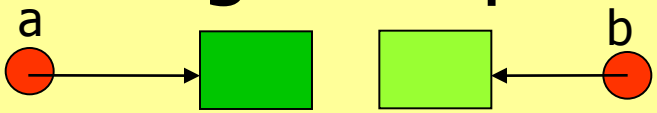


l'exécution de « a = b » renverra FAUX

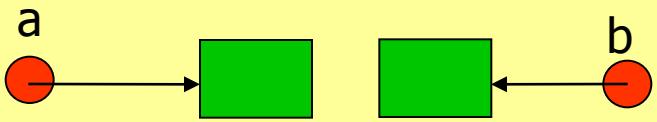


l'exécution de « a = b » renverra VRAI

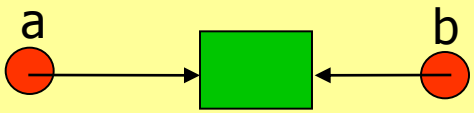
- estEgal : l'opération d'égalité de contenu



l'exécution de « a.estEgal(b) » renverra FAUX



l'exécution de « a.estEgal(b) » renverra VRAI



l'exécution de « a.estEgal(b) » renverra VRAI

Les structures de contrôle

- Un algorithme s'exécute instruction par instruction dans l'ordre annoncé dans le corps de l'algorithme

- Cette exécution linéaire a trois exceptions
  - l'instruction est une structure de contrôle conditionnelle
  - l'instruction est une structure de contrôle itérative
  - l'instruction est un appel d'opération

Les structures de contrôle conditionnelle (1)

- Si condition Alors

  liste d'instructions Lok

  FinSi

où condition est une expression booléenne.

- La liste d'instructions LOk sera exécutée que si le résultat de la condition est vraie.
- Dans tous les cas, l'algorithme se continue en séquence à partir de l'instruction qui suit le mot clé Fin.

Les structures de contrôle conditionnelle (2)

- Si condition Alors
  liste d'instructions Lok
  Sinon
  liste d'instructions LPasOk
  FinSi

où condition est une expression booléenne.

- La liste d'instructions LOK sera exécutée si le résultat de la condition est vrai et la liste d'instructions LPasOK si le résultat de la condition est faux.

- Dans tous les cas, l'algorithme se continue en séquence à partir de l'instruction qui suit le deuxième mot clé Fin.

Les structures de contrôle itérative (1)

- TantQue condition Faire
  liste d'instructions LOk
  FinTantQue

où condition est une expression booléenne.

- La liste d'instructions LOK sera exécutée tant que le résultat de la condition est vrai.

- Dès que le résultat de la condition est faux, l'algorithme reprend en séquence à partir de l'instruction qui suit le mot clé FinTantQue.

Les structures de contrôle itérative (2)

- Faire
  liste d'instructions Lok
  TantQue condition

où condition est une expression booléenne.

- La liste d'instructions LOK est exécutée une première fois puis autant de fois que le résultat de la condition est vrai.

- Dès que le résultat de la condition est faux, l'algorithme reprend en séquence à partir de l'instruction qui suit le mot clé TantQue.

Le type abstrait Entree (1)

- Tous les programmes ont besoin de récupérer des informations venant de l'extérieur.
- En général, ces informations proviendront de l'utilisateur qui exécute le programme
- Cet utilisateur donne ces informations via des périphériques d'entrées comme un clavier, une souris, un lecteur de codes à barre, ….

Le type abstrait Entree (2)

- Dans notre approche de l'algorithmique, un périphérique d'entrée sera un objet du type abstrait Entree

- En effet, tous ces objets ont en commun les opérations de lecture qu'ils peuvent faire

- Ils sont différents dans leurs structures (en mémoire et composants physiques). Mais la structure d'un objet ne relève pas de l'agorithmique.

Le type abstrait Entree (3)

- TYPE ABSTRAIT Entrée
  - Concept :
    Ce type permet de modéliser un périphérique d'entrée (ex: clavier, lecteur code à barre, etc) et de permettre l'affichage de l'information saisie.

  - Opérations de base
    Constructeur Entree : creerEntree(Chaine localisationPeripherique) : Entrée
    Transformateur Entree : lireChaine(Chaine) : Entrée
    Transformateur Entree : lireCaractere(Caractere) : Entrée
    Transformateur Entree : lireBooleen(Booleen) : Entrée
    Transformateur Entree : lireEntier(Entier) : Entrée
    Transformateur Entree : lireReel(Reel) : Entree

Le type abstrait Sortie (1)

- Tous les programmes ont besoin d'éditer des informations vers l'extérieur.

- Ces informations pourront s'adresser à l'utilisateur (via son écran) ou à d'autres périphériques (imprimantes, …)

Le type abstrait Sortie (2)

- Dans notre approche de l'algorithmique, un périphérique d'entrée sera un objet du type abstrait Entree

- En effet, tous ces objets ont en commun les opérations de lecture qu'ils peuvent faire

- Ils sont différents dans leurs structures (en mémoire et composants physiques). Mais la structure d'un objet ne relève pas de l'agorithmique.

Le type abstrait Sortie (3)

- TYPE ABSTRAIT Sortie
  - Concept
    Ce type permet de modéliser un périphérique de sortie (ex: écran, haut-parleur, etc...)

  - Opérations de base
    Constructeur Sortie : creerSortie (Chaine localisationPeripherique) : Sortie
    Transformateur Sortie : ecrireChaine (Chaine) : Sortie
    Transformateur Sortie : ecrireCaractere (Caractere) : Sortie
    Transformateur Sortie : ecrireBooleen (Booleen) : Sortie
    Transformateur Sortie : ecrireEntier (Entier) : Sortie
    Transformateur Sortie : ecrireReel (Reel) : Sortie
    Transformateur Sortie : ecrireNL () : Sortie

  - Opérations d'extension
    Transformateur Sortie : ecrireChaineNL (Chaine) : Sortie
    Transformateur Sortie : ecrireCaractereNL (Caractere) : Sortie
    Transformateur Sortie : ecrireBooleenNL (Booleen) : Sortie
    Transformateur Sortie : ecrireEntierNL (Entier) : Sortie
    Transformateur Sortie : ecrireReelNL (Reel) : Sortie