Technologie Objet (TOB)

Xavier Crégut...
<Prénom.Nom@enseeiht.fr> ou < *nom@n7.fr* >

TD	Enseignant
AB	Xavier Crégut
CD	Judicael Bedouet
EF	Meriem Ouederni
GH	Xavier Crégut
IJ	Judicael Bedouet
KL	Guillaume Dupont

TP	Enseignant
A	Louis Bonnet
B	Neeraj Singh
C	William Charles
D	Simone Gasparini
E	Meriem Ouederni
F	Jean-Claude Lebegue

TP	Enseignant
G	Estelle Chigot
H	Simone Gasparini
I	Marie Pelissier
J	Anaël Megna
K	Guillaume Dupont
L	Peter Rivière

Objectif : Programmation objet illustrée avec Java et UML

Prérequis :

1. Programmation Impérative (PIM) : TOB est la suite de PIM
2. Langage C : pour les aspects syntaxiques (Java s'inspire de la syntaxe de C)

Plan de cette présentation

Organisation du module

7 Cours : asynchrones, non programmés à l'EDT (sauf le 1er)

- But : présenter les concepts (en général, illustrés sur des exemples concrets)
- Votre objectif : **comprendre** les notions présentées en cours
- Moyen : questionnaires Moodle + poser des questions sur Moodle (forum) ou Discord

9 Travaux Dirigés (TD) :

- Assimiler les concepts du cours : savoir quand, comment et pourquoi les utiliser
- Le but est de **prendre du recul**
- Un corrigé rédigé des TD sera fourni (mais soyez présents et actifs en TD !)
- Commence par **QCM de 5'** : 1) cours lu ? + 2) ponctualité + 3) jusqu'à 2 points bonus sur l'examen

10 Travaux Pratiques (TP) : parfois application du cours, sans TD préparatoire !

- Travail sur machine, en autonomie.
- Le chargé de TP est là pour vous aider, pas pour vous donner la solution
- Fichiers fournis et à rendre via GIT : pousser le travail à la fin de chaque question !
- Pas de note : activité volontaire... mais rentable (à court et long terme !)
- Un corrigé sous forme des programmes attendus sera fourni (et peut-être un corrigé rédigé)

4 Séances Projet : pour deux petits projets (mini-projet et projet court)

1 Projet Long : réalisé conjointement avec le cours Méthodes Agiles (Gestion de projet)

Petits projets : mini-projet (PR01) et projet court (PR02)

Principe :

1. **Réalisation du projet :**
   - 2 séances prévues pour réaliser le projet, encadrées par 1 enseignant / groupe de TD
   - des tests automatiques sont lancés de temps en temps sur le code poussé sur GIT
     - But : vous aider à rendre un meilleur projet
2. **Premier rendu : rendre un projet complet !**
3. **Évaluation du premier rendu via :**
   1) un retour global
   - à vous de voir ce qui vous concerne
   - permet de connaître les principales remarques faites (même si non concerné)

   2) un retour de l'enseignant de TP sur les aspects très spécifiques de votre projet

   Attention : Cette première version n'est pas notée mais si des parties ne sont pas traitées, des points de pénalité peuvent être appliqués !
4. **Deuxième rendu :** tenir compte des retours pour améliorer le projet
5. **Retour final et note :** automatiquement et par l'enseignant de TP (retour sur GIT)

Mini-projet (PR01) : Modularité et test (dès la première semaine)

Projet court (PR02) : Concepts objets (mi-février à mars).

Double correction $\Longrightarrow$ travail important pour enseignants $\Longrightarrow$ **Jouez le jeu et ne trichez pas !**

Projet long

Objectif : Réaliser un projet ambitieux en équipe en appliquant une méthode agile (SCRUM).

Les équipes :

- 6 à 8 personnes du même groupes de TD (à centraliser par le délégué)
- 4 équipes par groupes de TD
- Le suivi est fait par l'enseignant de TD TOB

Le sujet : choisi par l'équipe

- Le sujet doit être ambitieux et original
- But : appliquer Méthodes Agiles et TOB pour en traiter une partie
- L'application devra avoir une interface utilisateur graphique

Déroulement :

1. Constitution des équipes
2. Proposition d'au moins 4 sujets par équipe
3. Sélection d'un sujet par l'enseignant de TD TOB
4. Description détaillée des fonctionnalités de l'application
5. Travail avec Gestion de projet (voir page suivante)

Charge étudiant :

- environ 40 h par étudiant : définition du sujet + 3 itérations de 2 semaines à 6h/semaine

Gestion de projet / Méthodes Agiles (échéancier projet long)

Cours Gestion de projet / Méthodes agiles :

- Gilles FRANÇOIS (Thales) et Olivier DUFFORT (Sopra Stéria)
- Appliqué sur le projet TOB, suivant le planning ci-après
- **Une opportunité pour vous ! Saisissez là !**

Planning :

- S04 : **Jalon** "Constitution des équipes"
- S05 : ...semaine de ski...
- S06 : **Jalon** "Proposition de sujets"
- S07 : (12/2) **CM SCRUM 1**
  + **Jalon** "Validation d'un sujet"
- S08 : **Jalon** "Description détaillée du sujet" (fonctionnalités)
- S09 : (26/2) **CM SCRUM 2** (Vision)
- S10 : (04/3) **TD SCRUM 1** (Backlog)
- S11 :
- S12 : (18/3) **TD SCRUM 2** (Sprint planning)
- S13 : — Itération 0 1/1
- S14 : — Itération 1 1/2
- S15 : Vacances
- S16 : Vacances
- S17 : — Itération 1 2/2
- S18 : — Itération 2 1/2 +
  (29/4) **TD SCRUM 3** (Rétrospective)
- S19 : — Itération 2 2/2
- S20 : — Itération 3 1/2
- S21 : — Itération 3 2/2
- S22 : (28/5) **Oral projet** TOB/SCRUM

Exemples de sujets

1. Assistant d'aide aux raffinages.
2. Application de proposition de repas
3. Gestion et maintenance d'une maison intelligente
4. Jeu guidé pour apprendre la programmation
5. Manuscript Manager (aide à l'écriture d'un roman)
6. Réalisation d'un RayTracer
7. Simulation de la propagation d'un virus
8. Escape game
9. Application de gestion des crèches
10. Programme d'apprentissage musical

Interdit :

- Pas d'application distribuée, client-serveur (cours en 2A)
- Pas de sujet type « plusieurs morceaux indépendants » (exemple : plusieurs petits jeux)

À éviter :

- Beaucoup de données, peu de traitements : Gestion d'un hôtel, d'une agence de voyage, d'une bibliothèque, réservation de vols, etc.

Choisissez un sujet qui vous intéresse, original, ambitieux (tout ne sera pas traité).

Évaluation

Examen écrit : 50 %

- 1 heure 30, actuellement programmé le **13 mai 2024 à 10h**
- sans document, feuille A4 autorisée, à rendre avec la copie
- Annales disponibles sur la page du module (examen 2017 avec corrigé)

Mini-Projet : 15 %

Projet Court : 15 %

Projet Long : 20 % (aussi évalué en **Gestion de projet** sur l'UE SHS)

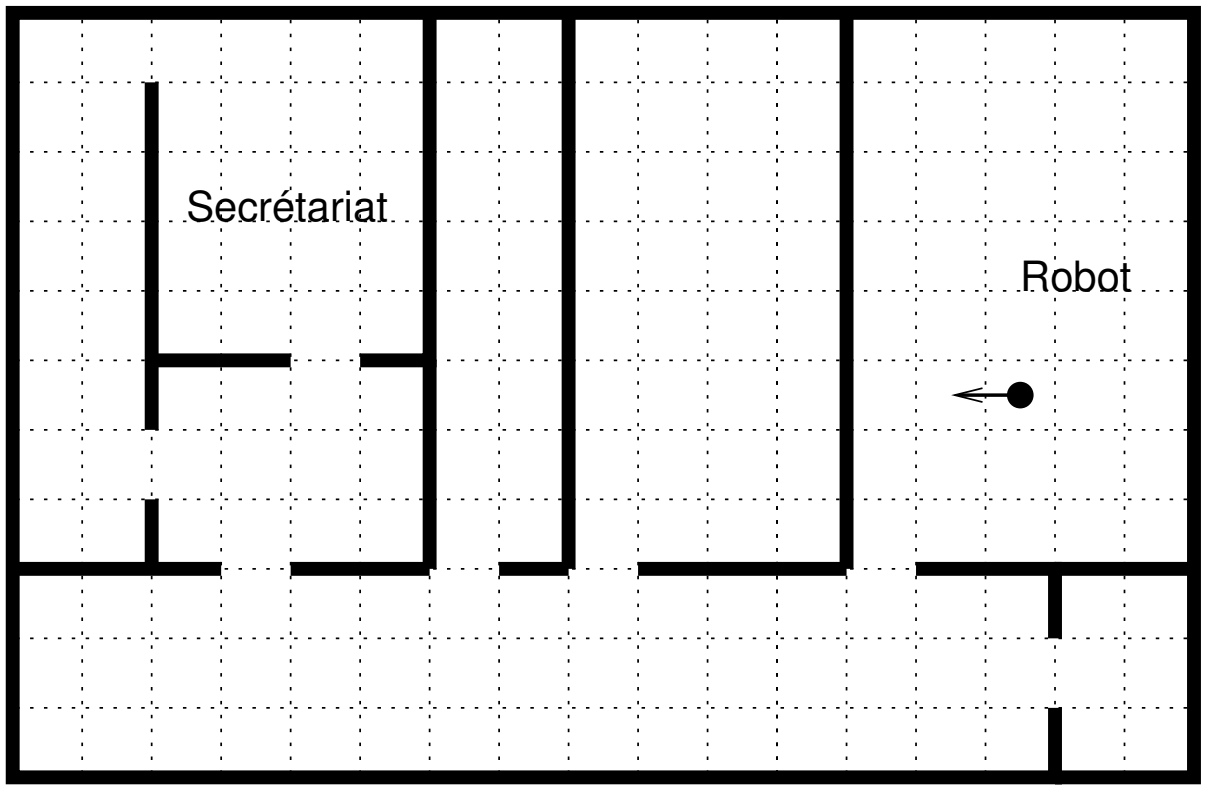
- rendus intermédiaires, rapport écrit, code
- Oral : 1) démonstration, 2) présentation technique et 3) gestion de projet
- Les orateurs seront tirés au hasard au début de la présentation
- Quelques points attribués par l'équipe

Session 2 :

- Examen : examen écrit (ou oral)
- Petits projets : Mêmes sujets ou nouveaux sujets (avec oral)
- Projet long : Même projet (avec oral)

Modéliser un robot

Exercice 1 Modéliser un robot capable d'avancer d'une case et de pivoter de 90° vers la droite. On pourra alors le guider de la salle de cours (position initiale du robot) jusqu'au secrétariat.



Types et sous-programmes associés

En pseudo-code

1. On sait définir un type Robot :

```
Type RobotType1 =
   Enregistrement
      x: Entier; -- abscisse
      y: Entier; -- ordonnée
      direction: Direction
   FinEnregistrement

Type Direction = (NORD, EST, SUD, OUEST)
```

2. On sait modéliser ses opérations :

```
Procédure avancer(r: in out RobotType1)
   -- faire avancer le robot r
  Début
   ...
  Fin

Procédure pivoter(r: in out RobotType1)
   -- faire pivoter le robot r
   -- de 90° à droite
  Début
   ...
  Fin

Procédure initialiser(r: out RobotType1
      x, y: in Entier, d: in Direction)
   -- initialiser le robot r...
  Début
   r.x <- x
   r.y <- y
   r.direction <- d
  Fin
```

3. On sait utiliser des robots :

```
   Variable
      r1, r2: RobotType1;
   Début
      initialiser(r1, 4, 10, EST)
      initialiser(r2, 15, 7, SUD)
      avancer(r1)
      pivoter(r2)
   Fin
```

En général, 1 + 2 définit un module et 3 un utilisateur du module (application, module...)

Version objet

1. Modéliser les robots :

Classe = Données + Traitements

RobotType1
x : Entier y : Entier direction : Direction
avancer pivoter
initialiser(**in** x, y : int, **in** d : Direction)

Notation UML :

Nom de la classe	
attributs	*(état)*
opérations	*(comportement)*
constructeurs	*(initialisation)*

2. Utiliser les robots (pseudo-code)

```
Variable
   r1, r2: RobotType1
Début
   r1.initialiser(4, 10, EST)
   { r1.x == 4 }
   r2.initialiser(15, 7, SUD)
   r1.pivoter
   r2.avancer
   -- r1 ou r2, paramètre privilégié
Fin
```

r1 : RobotType1
x = 4 y = 10 direction = ~~EST~~ SUD
avancer pivoter
initialiser(**in** x, y : int, **in** d : Direction)

(objet r1)

r2 : RobotType1
x = 15 y = ~~7~~ 6 direction = SUD
avancer pivoter
initialiser(**in** x, y : int, **in** d : Direction)

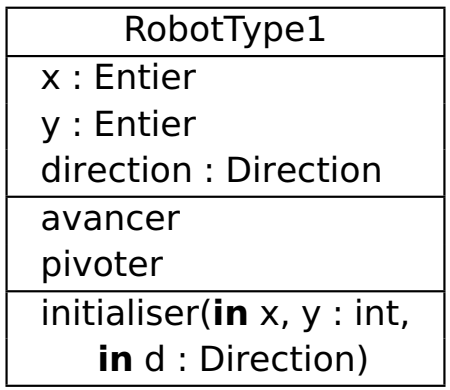
(objet r2)

Rq : initialiser correspond à un constructeur : permet d'initialiser un objet lors de sa création.

$\Longrightarrow$ Tout objet créé est initialisé (comme souhaité pas son créateur)

D'UML à Java

Le diagramme UML donne le squelette de la classe Java




```
class RobotType1 {
   int x; // abscisse de ce robot
   int y; // ordonnée de ce robot
   Direction direction;

   RobotType1(int x, int y, Direction d) {
      ...
   }

   void avancer() {
      ...
   }

   void pivoter() {
      ...
   }
}
```

- Données et traitements sont tous dans la classe
- Le paramètre de type RobotType1 a disparu : **paramètre implicite** (**this**)...
- En Java, un **constructeur** porte le même nom que la classe et n'a pas de type de retour
- Une **classe**, ici RobotType1, définit un **type** (on peut déclarer des variables)...
  mais aussi un **module** (contient des sous-programmes...)

Le code est proche de C

```
class RobotType1 {
   int x; // abscisse de ce robot
   int y; // ordonnée de ce robot
   Direction direction;

   RobotType1(int x, int y, Direction d) {
      this.x = x;
      this.y = y ;
      this.direction = d;
   }

   void avancer() {
      // Déplacer suivant l'axe des X
      if (this.direction == Direction.OUEST) {
         this.x--;
      } else if (this.direction == Direction.EST) {
         this.x++;
      }

      // Déplacer suivant l'axe des Y
      if (this.direction == Direction.SUD) {
         this.y--;
      } else if (this.direction == Direction.NORD) {
         this.y++;
      }
   }

   void pivoter() {
      switch (this.direction) {
         case NORD: this.direction = Direction.EST; break;
         case EST: this.direction = Direction.SUD; break;
         case SUD: this.direction = Direction.OUEST; break;
         case OUEST: this.direction = Direction.NORD; break;
      }
   }
}
```

- déclaration des variables
- structures de contrôle
- opérateurs
- **this** nomme le paramètre implicite

```
class ProgrammeRobotType1 {
   public static void main(String[] args) {
      // Créer deux robots
      RobotType1 r1 = new RobotType1(4, 10,
            Direction.EST);
      RobotType1 r2 = new RobotType1(15, 7,
            Direction.SUD);

      // Manipuler les robots
      r1.pivoter();
      r2.avancer();

      // Afficher quelques caractéristiques
      System.out.println("r1.direction = "
            + r1.direction);
      System.out.println("r2.y = " + r2.y);

      assert r1.direction == Direction.SUD;
      assert r2.y == 6;
   }
}
```

Et les types privés ?

Moyen : Pour chaque membre d'une classe, on peut préciser un droit d'accès : public, privé, etc.

Solution : Déclarer les attributs privés (accessibles seulement depuis la classe).

Conséquence : Définir des accesseurs publics pour obtenir x, y et la direction souvent nommés getXxx() en Java

```
public class RobotType1 {
   private int x; // abscisse de ce robot
   private int y; // ordonnée de ce robot
   private Direction direction;

   public RobotType1(int x, int y, Direction d) {
      ...
   }

   public int getX() {
      return this.x;
   }

   public int getY() {
      return this.y;
   }

   public Direction getDirection() {
      return this.direction;
   }

   public void avancer() {
      ...
   }

   public void pivoter() {
      ...
   }
}
```

Améliorer le code de avancer et pivoter ?

- **Règle :** Limiter le recours aux conditionnelles car souvent elles sont un frein à l'extensibilité !
- Pour **Avancer**, on ajoute à x (ou à y) soit 0, 1 ou -1 suivant la direction
  - **Idée :** stocker cet entier dans un tableau que l'on appelle dx (et dy)
  - Il n'est pas propre à un objet RobotType1 mais partagé par tous
    $\Longrightarrow$ C'est un attribut de classe (**static**) et non un attribut d'instance (comme x, y ou direction)
- **pivoter :** On peut exploiter les propriétés des types énumérés de Java

```
public class RobotType1 {
   private int x; // abscisse de ce robot
   private int y; // ordonnée de ce robot
   private Direction direction;

   private static int[] dx = {0, 1, 0, -1}; // déplacement suivant direction
   private static int[] dy = {1, 0, -1, 0};

   public RobotType1(int x, int y, Direction d) {
      this.x = x;
      this.y = y ;
      this.direction = d;
   }

   public void avancer() {
      this.x += RobotType1Mieux.dx[this.direction.ordinal()];
      this.y += RobotType1Mieux.dy[this.direction.ordinal()];
   }

   public void pivoter() {
      int position = this.direction.ordinal();
      position = (position + 1) % Direction.values().length;
      this.direction = Direction.values()[position];
   }
}
```

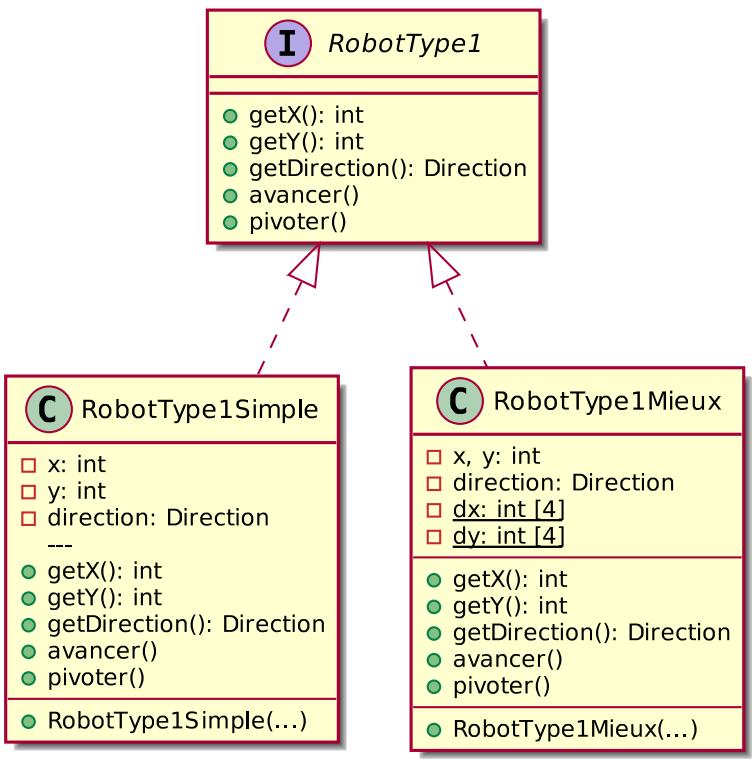
Interfaces

Le problème : On a écrit deux versions de RobotType1 : une avec des conditionnelles, l'autre sans. Comment garder les deux ?

Solution : On leur donne des noms différents : RobotType1Simple et RobotType1Mieux

Problème : Comment écrire du code qui marche avec l'une et l'autre (p.ex. faire suivre un carré à un robot) ?

Solution : Spécifier les opérations disponibles sur un RobotType1, grâce à une interface.




```
public interface RobotType1 {
   void avancer();
   void pivoter();
   int getX();
   int getY();
   Direction getDirection();
}
```

Une interface ne contient que des spécifications de méthodes, pas de code (jusqu'à Java7) !
La documentation est essentielle (même si omise ici) !

```
public class RobotType1Simple implements RobotType1 {
   ...
}

public class RobotType1Mieux implements RobotType1 {
   ...
}
```

RobotType1Simple et RobotType1Mieux sont des sous-types de RobotType1

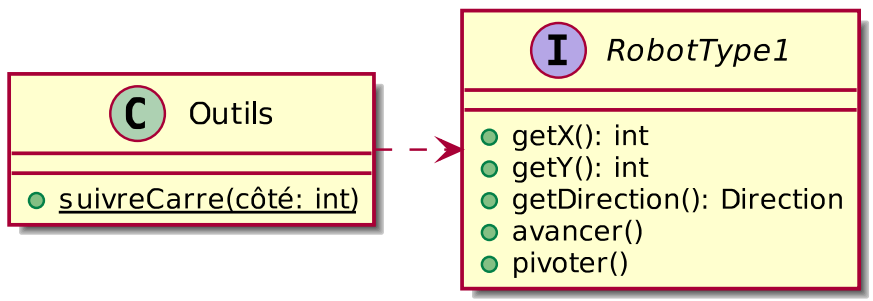
Interfaces : intérêt, sous-typage, liaison dynamique

Intérêt : Pouvoir écrire du code qui s'appuie sur l'interface et pourra s'appliquer sur tout sous-type de l'interface.

```
public class Outils {

   public static void suivreCarre(RobotType1 robot, int cote) {
      for (int numeroCote = 0; numeroCote < 4; numeroCote++) {
         // progresser de cote cases
         for (int nbCases = 0; nbCases < cote; nbCases++) {
            robot.avancer();
         }

         robot.pivoter();
      }
   }
}
```



Question : Est-ce que ce code est valide ?
OUI : liaison statique (compilateur)
Le compilateur vérifie que chaque appel correspond bien à une opération (signature).

Exemple : L'appel `robot.avancer()` est valide car il existe bien dans `RobotType1` (type de `robot`) une opération (signature) `avancer()` (sans paramètre).

Question : Quelle méthode « avancer » (ou « pivoter ») sera exécutée ?

Principe de substitution et liaison dynamique

Exemple d'utilisation de la classe Outils :

```
public class ExempleOutils {

   public static void main(String[] args) {
      RobotType1Simple r1 = new RobotType1Simple(3, 2, Direction.NORD);
      RobotType1Mieux r2 = new RobotType1Mieux(15, 0, Direction.EST);

      Outils.suivreCarre(r1, 5);
      Outils.suivreCarre(r2, 3);

      assert r1.getX() == 3;
      assert r1.getY() == 2;
      assert r1.getDirection() == Direction.NORD;

      assert r2.getX() == 15;
      assert r2.getY() == 0;
      assert r2.getDirection() == Direction.EST;
   }
}
```

Principe de substitution : Là où un type est attendu, un objet d'un sous-type peut être utilisé.

$\Longrightarrow$ On peut appeler suivreCarre avec un objet de type RobotType1Simple ou RobotType1Mieux

Liaison dynamique : Le choix de la méthode à exécuter se fait à l'exécution (pas à la compilation).

- La méthode est dans la classe de l'objet sur lequel la méthode est appelée.

Principe de substitution + liaison dynamique $\Longrightarrow$ extensibilité

- suivreCarre, prévue pour RobotType1, fonctionne avec RobotType1Simple, RobotType1Mieux...

RobotMetreur : mesurer la distance parcourue

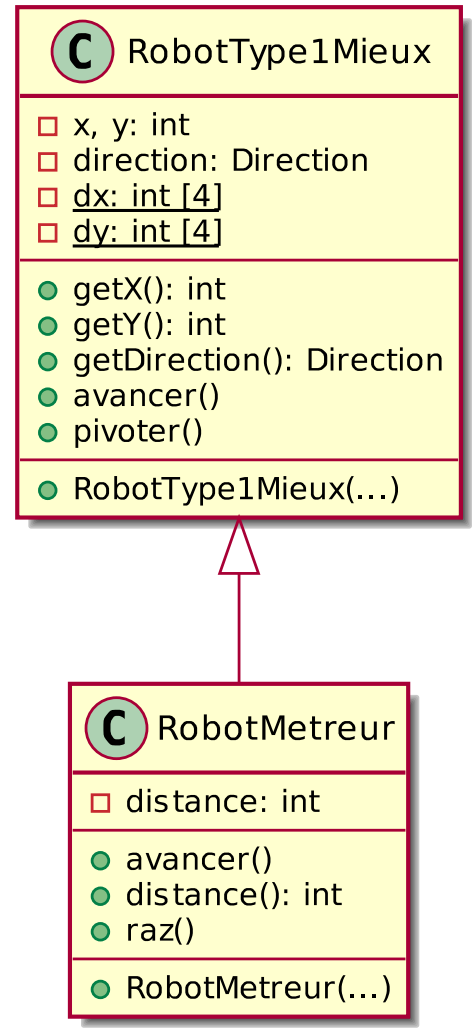
Objectif : Définir un robot métreur qui peut avancer et pivoter comme un robot type 1 mais qui en plus enregistre la distance parcourue. On peut remettre à zéro la distance parcourue.

Question : Comment faire ?

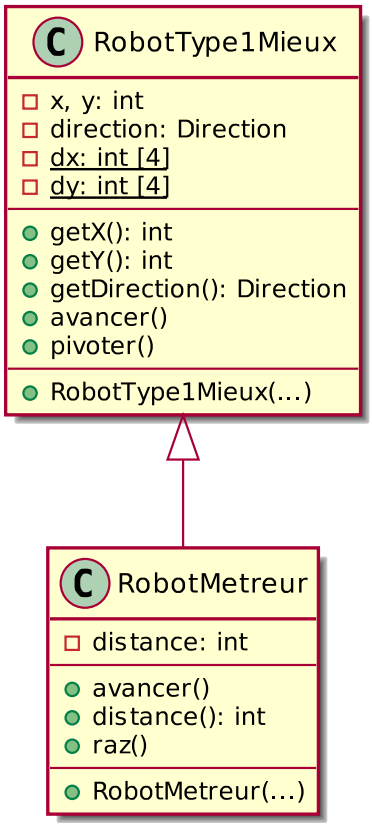
Solution :

- Un robot métreur ressemble à un robot type 1.
- Il a le même comportement. En plus il enregistre une distance.
- Un robot métreur est un sous-type de robot type 1

$\Longrightarrow$ On peut le définir en utilisant l'héritage (spécialisation en UML)



La classe RobotMetreur




```java
public class RobotMetreur extends RobotType1Mieux {

   private int distance;

   public RobotMetreur(int x, int y, Direction d) {
      super(x, y, d);
      this.distance = 0;
   }

   public int distance() {
      return this.distance;
   }

   @Override public void avancer() {
      super.avancer();
      this.distance += 1;
   }

   public void raz() {
      this.distance = 0;
   }

}
```

Intérêt :

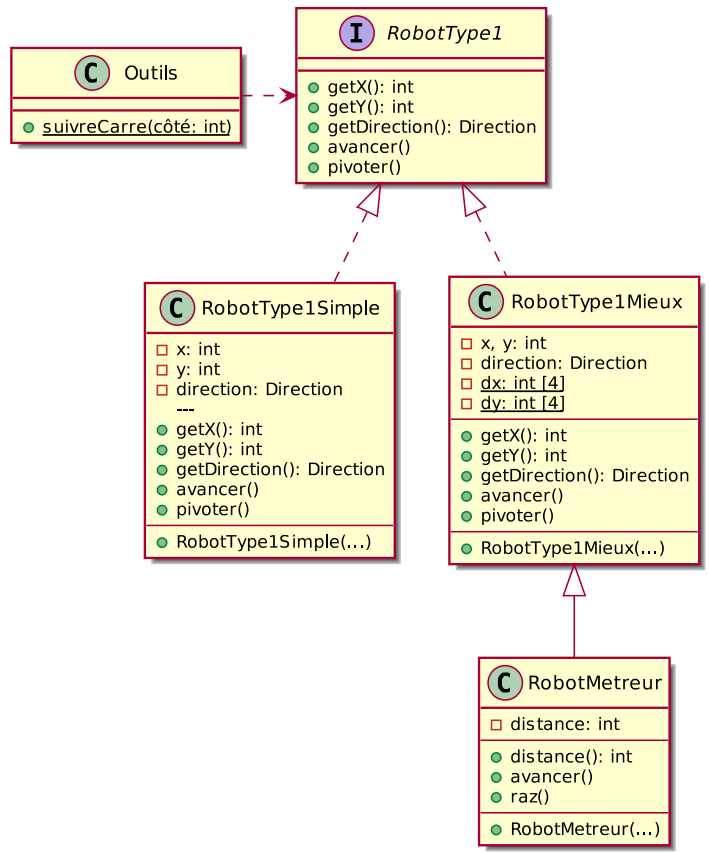
- Le code de la classe RobotType1Mieux n'est pas dupliqué (il est hérité).
- Mais peut être redéfini (par exemple : la méthode avancer)
- Réalise le principe ouvert/fermé (ouvert à l'extension, fermé à la modification)

RobotMetreur est un RobotType1 : extensibilité

```
public class ExempleRobotMetreur {

   public static void main(String[] args) {
      RobotMetreur r1 = new RobotMetreur(3, 2, Direction.NORD);
      Outils.suivreCarre(r1, 5);
      System.out.println("distance parcourue = " + r1.distance());

      RobotType1Mieux r2 = r1;
      r2.avancer();     // la distance augmente ?
      System.out.println("distance parcourue = " + r1.distance());
      r2.raz();         // possible ?
   }
}
```

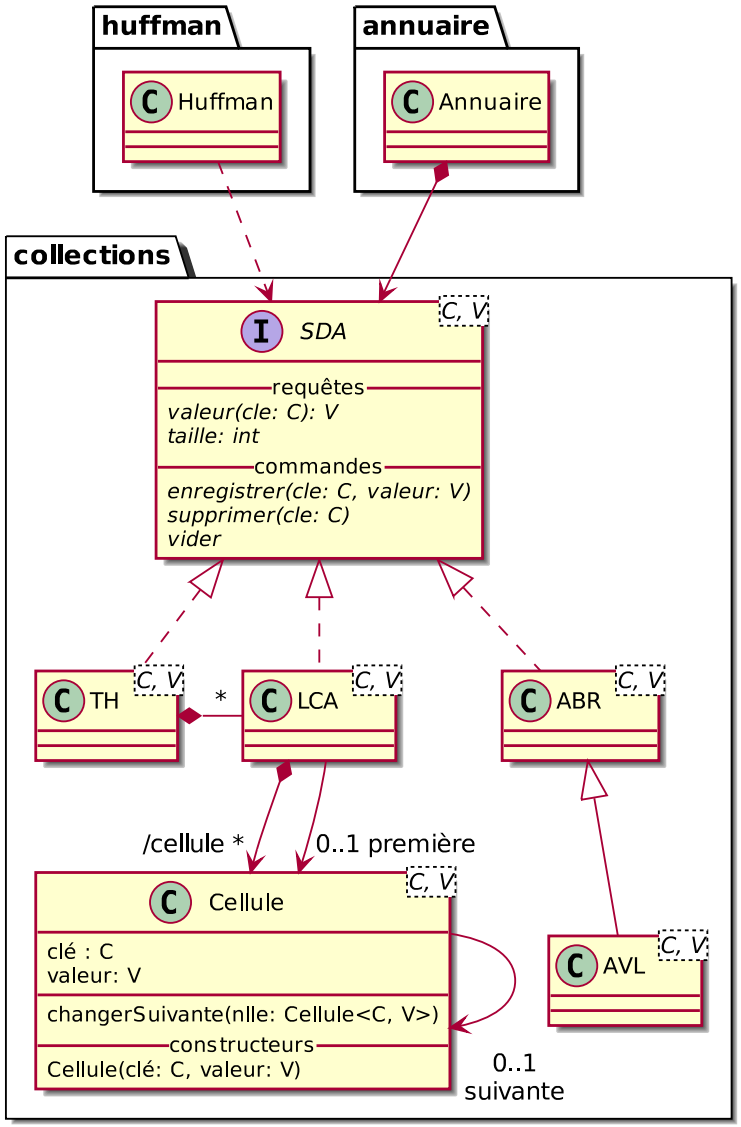


- La liaison dynamique est la même que pour les interfaces.
- Mais contrairement aux interfaces, la méthode a un code dans la super-classe et la sous-classe.
- Le code choisi sera bien celui de la classe de l'objet.

  r2.avancer() fera bien augmenter la distance (idem sur suivreCarre).
- **Extensibilité :** Tous les codes qui fonctionnaient avec un RobotType1Mieux, fonctionnent aussi avec RobotMetreur (extensibilité) sans avoir à connaître explicitement RobotMetreur.
- `r2.raz()` : impossible à cause de la liaison statique !

  L'appel est refusé par le compilateur car pas d'opération raz dans RobotType1Mieux.

Concepts objets appliqués aux SDA (structures de données associatives)



- **UML :** vue synthétique sur le système (ici architecture)
- **Classe :** équivalent d'un module (exemple Cellule)
- Plusieurs applications (annuaire, huffman...) ont besoin de structure de données associatives (SDA)
- Ces SDA sont spécifiées par une **interface** : SDA
- Plusieurs **réalisations** (classes) peuvent être définies : LCA, TH, ABR...
- **Sous-typage :** une LCA est une SDA, une TH est une SDA...
- **Principe de substitution :** Partout où on attend une SDA on pourra mettre un de ses sous-types (LCA, TH, ABR...)
- **Liaison dynamique :** Quand une application appellera une méthode d'une SDA, c'est la version dans la classe de l'objet (LCA, TH, ABR...) qui sera exécutée.
- **Héritage :** Spécialiser une classe (sous-type avec comportement adapté et complété) : exemple AVL.
- **Relations d'utilisation :** une classe utilise une autres classe : association (Cellule – Cellule), composition (TH – LCA)
- Maître mot : **extensibilité**

Thèmes traités dans l'UE TOB

1. Modularité et encapsulation : Classe
2. Relations entre classes
   Interfaces (sous-typage et liaison dynamique)
   Généricité
3. Héritage, classes abstraites
4. Exceptions
5. Patrons de conceptions (« solutions éprouvées à des problèmes récurrents »)
6. Structures de données
7. UML : Modélisation du comportement (séquence, machine à état, activité)
8. Interfaces graphiques et programmation événementielle

Suppression des cours magistraux en présentiel

Pourquoi supprimer les cours magistraux en présentiel ?

- J'y pense depuis plusieurs années : absentéisme, manque de concentration en cours, 1h45 c'est trop long, rythme trop rapide ou trop lent suivant les étudiants...
- Création du département Sciences du Numérique avec, pour chaque UE, limite du nombre d'heures étudiant (30 séances) et du coût (nombre d'heures enseignant, $\sim$ 420 hEqTD)

$\Longrightarrow$ L'occasion (obligation !) de franchir le pas !

Conséquences sur les cours ?

- Des supports de cours, avec plus d'explications et la correction des principaux exercices.
- Questionnaires Moodle / petits exercices pour valider que le cours a été compris.
- Des QCM en début de TD...
- Un forum sous Moodle pour poser des questions et le salon TOB de Discord...
- Des questions à vos enseignants de TP, TD, Cours !
- Séances de soutien/tutorat possibles ! À demander via les délégués de TD !
- **Rappel :** Seulement 23 créneaux au lieu de 30 à l'EDT (9 TD + 10 TP + 4 PR) sans compter 1 examen et 1 oral de projet
  $\Longrightarrow$ Ceci vous laisse du temps (7 séances) pour lire et comprendre le cours !