

Les classes

NFA032

Présentation n°2

Les classes

- Plan de présentation des notions NFA032
 - Notions de base (hors classes) : NFA031
 - Notion de classe
 - Variable d'instance (non statique)
 - Méthode, dont constructeur
 - Référence, passage de paramètre, effet de bord
 - Interface
 - Héritage
 - Récursion
 - Méthode récursive
 - Type récursif : exemple des listes
 - Exceptions

Terminologie

- Terme programmation objet / **Terme Java**
 - Même vocabulaire :
 - Classe
 - objet = instance
 - Attribut = champ = variable d'instance / **variable non statique**
 - Attribut commun (partagé) = variable de classe / **variable statique**
 - Fonction / **méthode statique**
 - Méthode / **méthode non statique**
 - envoi de message / **appel de méthode** ou **accès à une variable**
- Type enregistrement = classe sans méthode

Exemple : les points

- Ecrire le code Java permettant de :
 - Représenter des points du plan (coordonnées cartésiennes)
 - Afficher les informations d'un point
 - Créer un nouveau point à partir d'informations diverses
 - Réinitialiser les coordonnées d'un point
 - Déplacer un point selon une translation (Δ_x, Δ_y)

Les points

- Séance précédente : deux solutions présentées
 - Point = objet tableau
 - Point = objet d'un type enregistrement
 - Rappel de cette solution
- Nouvelle solution proposée
 - Point = objet d'une classe
 - Deux variantes, basées sur la conception des méthodes ne retournant pas de résultat direct (exemple : affichage)

Un point = un type enregistrement (1/2)

- Définitions en Java

```
class point {  
    double x,  
           y;  
}
```

```
// 1ère fonction constructeur : des coordonnées  
static point créer_point(double x, double y) {  
    point p = new point();  
    p.x = x; p.y = y;  
    return p;  
}
```

```
// 2ème fonction constructeur : un point existant (constructeur de recopie/clonage)  
static point créer_point(point p) { return créer_point(p.x, p.y); }
```

```
static void initialiser(point p, double x, double y) { p.x = x; p.y = y; }
```

```
static void afficher(point p) {  
    Terminal.ecrireString("( "+p.x+" , "+p.y+" )");  
}
```

```
static void déplacer(point p, double dx, double dy) { p.x += dx; p.y += dy; }
```

```
static void déplacer(point p, point d) { déplacer(p, d.x, d.y); }
```

Un point = un type enregistrement (2/2)

- Extrait du programme de test

```
Terminal.ecrireStringln("Petit jeu avec un point");
Terminal.ecrireStringln("=====");
Terminal.ecrireString("point P = ");
point P = créer_point(2, 3);
afficher(P);
Terminal.sautDeLigne();
double dx = -1, dy = 2;
Terminal.ecrireString("P est déplacé de (" + dx + ", " + dy + ") => point P = ");
déplacer(P, dx, dy);
afficher(P);
Terminal.sautDeLigne();
```

Manipulation 1 : les fonctions sont intégrées au type enregistrement

```
class point {
    double x,
           y;

    // fonctions de manipulation des points
    // =====

    // 1ère fonction constructeur : des coordonnées
    static point créer_point(double x, double y) {
        point p = new point();
        p.x = x; p.y = y;
        return p;
    }

    // 2ème fonction constructeur : un point existant (constructeur de copie/clonage)
    static point créer_point(point p) { return créer_point(p.x, p.y); }

    static void initialiser(point p, double x, double y) { p.x = x; p.y = y; }

    static void afficher(point p) {
        Terminal.ecrireString("( "+p.x+" , "+p.y+" )");
    }

    static void déplacer(point p, double dx, double dy) { p.x += dx; p.y += dy; }

    static void déplacer(point p, point d) { déplacer(p, d.x, d.y); }
```

Manipulation 1 : transformation du code du test

```
Terminal.afficheStringln("Petit jeu avec un point");
Terminal.afficheStringln("=====");
Terminal.afficheString("point P = ");
point P = point.créer_point(2, 3);
point.affiche(P);
Terminal.sautDeLigne();
double dx = -1, dy = 2;
Terminal.afficheString("P est déplacé de (" + dx + ", " + dy + ") => point P = ");
point.déplacer(P, dx, dy);
point.affiche(P);
Terminal.sautDeLigne();
```

- Remarque :
 - On peut effacer le `_point` dans le nom des fonctions constructeurs : `point.créer(...)`

Manipulation 2 : transformation en code objet pur

- Principe :

- Toute fonction f (méthode statique) d'une classe C ayant un paramètre p de type C est transformée en une méthode non statique, en effaçant p

`static R f(C p, reste)`

devient

`R f(reste)`

- Dans le corps original, remplacer p par `this`
- Tout appel à la fonction `f(e, reste)` est remplacé par l'envoi de message `e.f(reste)`
 - la valeur de e est une référence qui sera liée à `this` pour cet appel
 - l'objet référencé est appelé *l'objet receveur* du message
- Traitement spécial pour les constructeurs

- Autrement dit :

- Toute méthode d'une classe C a un paramètre implicite `this`, de type C , lequel sera lié à l'objet receveur d'un envoi de message.

Un point = une classe (1/3)

- Définition de la classe

```
public class point {  
    double x,  
        y;
```

Les constructeurs sont
expliqués après...

```
// 1er constructeur : des coordonnées  
point(double x, double y) {  
}  
  
// 2ème constructeur : un point existant (constructeur de copie/clonage)  
point(point p) {  
}  
  
// une méthode statique équivalente à créer_point de la version 0 ou 1  
static point créer(double x, double y) { return new point(x, y); }  
  
void initialiser(double x, double y) { this.x = x; this.y = y; }  
  
void afficher() { Terminal.ecrireString("( "+this.x+" , "+this.y+" )"); }  
  
void déplacer(double dx, double dy) { this.x += dx; this.y += dy; }  
  
void déplacer(point d) { this.déplacer(d.x, d.y); }  
}
```

Un point = une classe (2/3)

- Détail des constructeurs

```
public class point {  
  
    double x,  
           y;  
  
    // 1er constructeur : des coordonnées  
    point(double x, double y) {  
        this.x = x; this.y = y;  
        // pourrait être remplacé par this.initialiser(x, y);  
    }  
  
    // 2ème constructeur : un point existant (constructeur de recopie/clonage)  
    point(point p) {  
        this.x = p.x; this.y = p.y;  
        // pourrait être remplacé par this(p.x, p.y);  
        // pourrait être remplacé par this.initialiser(p.x, p.y);  
    }  
  
    // une méthode statique équivalente à créer_point de la version 0 ou 1  
    static point créer(double x, double y) {  
        return new point(x, y);  
    }  
  
    void initialiser(double x, double y) {  
        this.x = x; this.y = y;  
    }  
  
    void afficher() {
```

Un point = une classe (3/3)

- Code du test :

```
Terminal.ecrireStringln("Petit jeu avec un point");
Terminal.ecrireStringln("=====");
Terminal.ecrireString("point P = ");
point P = new point(2, 3);
P.afficher();
Terminal.sautDeLigne();
double dx = -1, dy = 2;
Terminal.ecrireString("P est déplacé de (" + dx + ", " + dy + ") => point P = ");
P.déplacer(dx, dy);
P.afficher();
Terminal.sautDeLigne();
```

Les constructeurs

- Un constructeur est une méthode
 - Appelée via l'opérateur **new** pour créer un nouvel objet ; **this** est lié à l'objet créé lors de l'exécution du corps du constructeur
 - Contraintes syntaxiques :
 - Porte le nom de la classe
 - Ne doit pas spécifier de type de retour
 - Ne contient aucune instruction **return**
 - Retourne toujours **this**, l'objet créé
 - Un constructeur peut en appeler un autre de sa classe
 - Appel de la forme **this** (...)
 - Cet appel doit impérativement figurer comme première instruction du corps du constructeur

Enchaînement des envois de messages

- Principe :
 - Plutôt que de définir une méthode ayant un résultat **void**, lui faire retourner l'objet receveur **this**, donc de type C : cela permettra de d'enchaîner les envois (envois en cascade)

Classe des points modifiée

- Modification de quelques méthodes :

```
// changement ici : le résultat 'void' est remplacé par 'point' => retourne l'objet receveur
point initialiser(double x, double y) {
    this.x = x; this.y = y;
    return this;
}

// changement ici : le résultat 'void' est remplacé par 'point' => retourne l'objet receveur
point afficher() {
    Terminal.ecrireString("( "+this.x+" , "+this.y+" )");
    return this;
}

// changement ici : le résultat 'void' est remplacé par 'point' => retourne l'objet receveur
point déplacer(double dx, double dy) {
    this.x += dx; this.y += dy;
    return this;
}

// changement ici : le résultat 'void' est remplacé par 'point' => retourne l'objet receveur
point déplacer(point d) { return this.déplacer(d.x, d.y); }
```

Classe des points modifiée

- Modification du programme de test :

```
Terminal.ecrireStringln("Petit jeu avec un point");  
Terminal.ecrireStringln("=====");  
Terminal.ecrireString("point P = ");  
point P = new point(2, 3).afficher(); // enchaînement de 2 messages, 'af:  
Terminal.sautDeLigne();  
double dx = -1, dy = 2;  
Terminal.ecrireString("P est déplacé de (" + dx + ", " + dy + ") => point P = ");  
P.déplacer(dx, dy).afficher(); // enchaînement de 2 messages  
Terminal.sautDeLigne();
```

Exercice : ajout des segments

- Compléter le code pour permettre la manipulation de segments :
 - Un segment est caractérisé par ses deux points extrémités
 - Les services attendus :
 - Affichage des caractéristiques d'un segment
 - Déplacement d'un segment selon une translation (Δ_x, Δ_y)
 - Calcul de la longueur d'un segment

Un segment = un enregistrement (1/2)

```
// 1ère fonction constructeur : coordonnées des extrémités
static segment créer_segment(double x1, double y1, double x2, double y2) {
    segment s = new segment();
    s.e1 = créer_point(x1, y1);
    s.e2 = créer_point(x2, y2);
    return s;
}
```

```
public class segment {
    point e1, // 1ère extrémité
           e2; // 2ème extrémité
}
```

```
// 2ème fonction constructeur : deux points extrémités
static segment créer_segment(point c1, point c2) {
    return créer_segment(c1.x, c1.y, c2.x, c2.y);
}

// 3ème fonction constructeur : un segment existant (recopie/clonage)
static segment créer_segment(segment s) {
    return créer_segment(s.e1, s.e2);
}

static void initialiser(segment s, double x1, double y1, double x2, double y2) {
    initialiser(s.e1, x1, y1);
    initialiser(s.e2, x2, y2);
}

static void afficher(segment s) {
    afficher(s.e1);
    Terminal.ecrireString("@");
    afficher(s.e2);
}
```

Un segment = un enregistrement (2/2)

- Suite des fonctions :

```
static void déplacer(segment s, double dx, double dy) {  
    déplacer(s.e1, dx, dy);  
    déplacer(s.e2, dx, dy);  
}  
  
static void déplacer(segment s, point d) { déplacer(s, d.x, d.y); }  
  
static double longueur(segment s) {  
    return Math.sqrt(Math.pow(s.e1.x-s.e2.x, 2)+Math.pow(s.e1.y-s.e2.y, 2)); }  
}
```

- Programme de test :

```
Terminal.ecrireStringln("Petit jeu avec un segment");  
Terminal.ecrireStringln("=====");  
Terminal.ecrireString("segment S = ");  
segment S = créer_segment(-1, 0, 3, 2);  
afficher(S);  
Terminal.ecrireStringln(" ; longueur = "+ longueur(S));  
dx = -1; dy = 2;  
Terminal.ecrireString("S est déplacé de (" +dx+" , "+dy+" ) => segment S = ");  
déplacer(S, dx, dy);  
afficher(S);  
Terminal.sautDeLigne();
```

Un segment = un objet (1/3)

- Définition de la classe :

```
public class segment {  
  
    point  e1,  // 1ère extrémité  
           e2;  // 2ème extrémité  
  
    // 1er constructeur : coordonnées des extrémités  
    segment(double x1, double y1, double x2, double y2) {  
        this.e1 = new point(x1, y1);  
        this.e2 = new point(x2, y2);  
    }  
  
    // 2ème constructeur : deux points extrémités  
    segment(point c1, point c2) {  
        this(c1.x, c1.y, c2.x, c2.y); // appelle le 1er constructeur  
    }  
  
    // 3ème constructeur : un segment existant (constructeur de copie/clonage)  
    segment(segment s) {  
        this(s.e1, s.e2); // appelle le 2ème constructeur  
    }  
  
    static segment créer(double x1, double y1, double x2, double y2) {  
        return new segment(x1, y1, x2, y2);  
    }  
}
```

Un segment = un objet (2/3)

- Suite du code :

```
// changement ici : le résultat 'void' est remplacé par 'segment' =>
segment initialiser(double x1, double y1, double x2, double y2) {
    this.e1.initialiser(x1, y1);
    this.e2.initialiser(x2, y2);
    return this;
}

// changement ici : le résultat 'void' est remplacé par 'segment' =>
segment afficher() {
    this.e1.afficher();
    Terminal.ecrireString("@");
    this.e2.afficher();
    return this;
}

// changement ici : le résultat 'void' est remplacé par 'segment' =>
segment déplacer(double dx, double dy) {
    this.e1.déplacer(dx, dy);
    this.e2.déplacer(dx, dy);
    return this;
}

// changement ici : le résultat 'void' est remplacé par 'segment' =>
segment déplacer(point d) {
    return this.déplacer(d.x, d.y);
}

double longueur() {
    return Math.sqrt(Math.pow(this.e1.x-this.e2.x, 2)
        +Math.pow(this.e1.y-this.e2.y, 2));
}
```

Un segment = un objet (3/3)

- Code du programme de test :

```
Terminal.ecrireStringln("Petit jeu avec un segment");
Terminal.ecrireStringln("=====");
Terminal.ecrireString("segment S = ");
segment S = new segment(-1, 0, 3, 2).afficher(); // enchaînement de 2 messages
Terminal.ecrireStringln(" ; longueur = "+ S.longueur());
dx = -1; dy = 2;
Terminal.ecrireString("S est déplacé de (" +dx+", "+dy+") => segment S = ");
S.déplacer(dx, dy).afficher(); // enchaînement de 2 messages
Terminal.sautDeLigne();
```

Autre exemple : les dates

- Voir
 - le support *introduction à la programmation objet en Java*
 - pages 26 à 31
- Les notions présentées dans ce support pages 1 à 31 sont dans le champ de cette séance.