

Algorithmique & Programmation

Créer ses propres types

yann.secq@univ-lille.fr

ABIDI Sofiene, ALMEIDA COCO Amadeu, BONEVA Iovka, CASTILLON Antoine,
DELECROIX Fabien, LEPRETRE Éric, Timothé ROUZÉ, SANTANA MAIA Deise,
SECQ Yann

Création de type

- Actuellement, nous utilisons des types « prédéfinis »: `int`, `boolean`, `String` ...
- Pour modéliser plus finement un problème, il serait intéressant de **pouvoir créer de nouveaux types**
- Notamment des types permettant d'**agréger des informations de nature différentes** au sein d'une même structure
- Cela permettrait de détecter des erreurs de typage à la compilation et d'améliorer la lisibilité de nos programmes
- Première étape vers la programmation orientée objets !

Définition d'un type

- Un type est défini par
 - **un nom**, qui correspond au nouveau type
 - **un ou plusieurs champs**, qui correspondent à des définitions de variables
- Une fois défini, le type est (ré)utilisable dans différents programmes

Exemple: Date

- Comment représenter une date (jour/mois/année)
- Utilisation possible d'un tableau `int[3]` ...
- ... ou création d'un nouveau type `Date` !
- Il faut **préciser le nom du type et sa constitution**, ie. les informations qui le définissent

Le type Date

Remarquez l'absence de « extends Program » !

```
class Date {  
    int jour;  
    int mois;  
    int année;  
}
```

← Nom du nouveau type

← Champs composant ce type

A sauvegarder dans un fichier nommé : Date.java

Utilisation du type Date

```
class TestDate extends Program {  
    void algorithm() {  
        Date today = new Date(); // création  
        today.annee = 2023; // accès à un champs  
        today.mois = 11; // et initialisation  
        today.jour = 6;  
        println(toString(today));  
    }  
    String toString(Date d) {  
        return d.jour + "/" + d.mois + "/" + d.annee;  
    }  
}
```

Un tableau de Date ?

```
class TestTableauDate extends Program {  
    Date newDate() { // bonne pratique !  
        Date d = new Date();  
        d.annee = (int) (random()*2023);  
        d.mois   = 1 + (int) (random()*12);  
        d.jour   = 1 + (int) (random()*31); // ok ...  
        return d;  
    }  
    void algorithm() {  
        Date[] dates = new Date[10];  
        for (int idx=0; idx<length(dates); idx++) {  
            dates[idx] = newDate();  
            println(toString(date[idx]));  
        }  
    }  
}
```

Un tableau de Date ?

```
class TestTableauDate extends Program {  
    Date newDate() { // bonne pratique !  
        Date d = new Date();  
        d.annee = (int) (random()*2023);  
        d.mois   = 1 + (int) (random()*12);  
        d.jour   = 1 + (int) (random()*31); // ok ...  
        return d;  
    }  
    void algorithm() {  
        Date[] dates = new Date[10];  
        for (int idx=0; idx<length(dates); idx++) {  
            dates[idx] = newDate();  
            println(toString(date[idx]))  
        }  
    }  
}
```

allocation d'un tableau

appel de la fonction créant et initialisant une date

Types hétérogènes

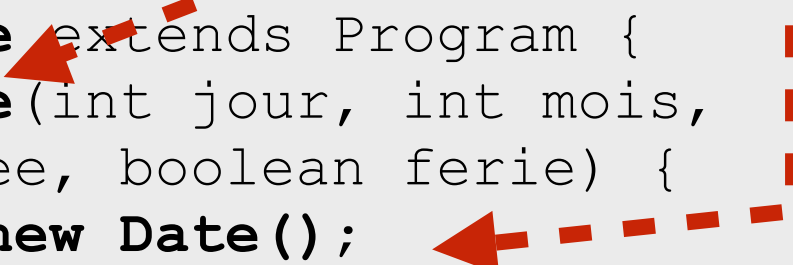
- `Date` est un type homogène : tous ses champs ont le même type (`int`)
- Rien n'empêche de définir un type hétérogène, c'est-à-dire dont les champs sont de différents types
- Comment représenter le fait qu'une date puisse être un jour férié ?

Le type Date

**ATTENTION A
CETTE NUANCE !**

```
class Date {  
    int jour;  
    int mois;  
    int annee;  
    boolean ferie;  
}
```

```
class TestDate extends Program {  
    Date newDate(int jour, int mois,  
        int annee, boolean ferie) {  
        Date d = new Date();  
        d.jour = jour; d.mois = mois;  
        d.annee = annee; d.ferie = ferie;  
        return d;  
    }  
    void algorithm() {  
        Date nouvelAn =  
            newDate(01, 01, 2023, true);  
        if (nouvelAn.ferie) {  
            println("Pas d'algo :(");  
        } else {  
            println("Chouette de l'algo :(");  
        }  
    }  
}
```




Les types énumérés

- Parfois, on a besoin de définir un ensemble de constantes qui constituent un tout
- Par exemple, les mois de l'année : janvier, février, mars, avril, mai, juin ...
- Dans cette situation, il est possible de définir une **énumération**
- Cela définit un type constitué d'un ensemble de constantes

Les types Mois et Date

ATTENTION : ce n'est plus class dans ce cas !



```
enum Mois {  
    JANVIER, FEVRIER, MARS, AVRIL, MAI,  
    JUIN, JUILLET, AOÛT, SEPTEMBRE,  
    OCTOBRE, NOVEMBRE, DECEMBRE;  
}
```

```
class Date {  
    int jour;  
    Mois mois;  
    int annee;  
    boolean ferie;  
}
```

Usage d'une énumération

```
class TestTableauDate extends Program {  
    Date newDateAleatoire() {  
        Mois[] année = new Mois[]{Mois.JANVIER,  
Mois.FEVRIER, ..., Mois.DECEMBRE};  
        Date d = new Date();  
        d.annee = (int) (random()*2022);  
        d.mois = année[1 + (int) (random()*12)];  
        d.jour = 1 + (int) (random()*31);  
        return d;  
    }  
    void algorithm() {  
        Date[] dates = new Date[10];  
        for (int idx=0; idx<length(dates); idx++) {  
            dates[idx] = newDateAleatoire();  
            println(toString(date[idx]));  
        }  
    }  
}
```

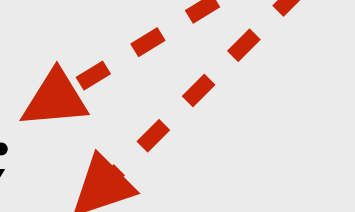
Un type dans un type ?

- Une fois un nouveau type défini, il est possible de le (ré)utiliser pour définir d'autres types !
- Prenons l'exemple d'une personne que l'on pourrait caractériser par
 - un nom et un prénom (`String`)
 - une date de naissance et de décès (`Date`)
 - et un code INSEE (`long`)
- Le type `Personne` réutilise le type `Date` !

Le type `Personne`

Il est possible de donner des valeurs par défaut aux champs

```
class Personne {  
    long insee = 0;  
    String nom = "?";  
    String prenom; // == null  
    Date    naissance;  
    Date    deces;  
}
```



Utilisation de Personne

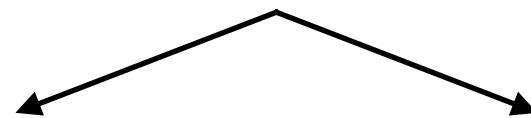
```
class TestPersonne extends Program {  
    Date newDate(int jour, int mois, int annee) { ...  
    Personne newPersonne(long insee, String nom,  
        String prenom, Date naissance, Date deces) {  
        Personne p = new Personne();  
        p.insee = insee; p.nom = nom; p.prenom = prenom;  
        p.naissance = naissance; p.deces = deces;  
        return p;  
    }  
    String toString(Personne p) {  
        return p.prenom+" "+p.nom+" (" +  
            toString(p.naissance)+"-"+toString(p.deces);  
    }  
    void algorithm() {  
        Personne p = newPersonne(1, "Turing", "Alan",  
            newDate(23,06,1912), newDate(07,06,1954));  
        println(toString(p));  
    }  
}
```


Synthèse

- Il est possible de **créer des types adaptés** au problème traité
- Les types sont définis par un nom et un ensemble de champs
- **Les types peuvent être homogènes** (champs de même type) **ou hétérogènes** (champs de types différents)
- **La création d'une valeur d'un type est similaire à l'allocation d'un tableau**
- On accède à un champ d'un type grâce la notation pointée: d.jour
- Il est possible de créer des tableaux de types que l'on a créé
- La création de type peut être (très) pratique pour votre projet ;)

Créer ses propres types

Le nouveau type est-il variable ou est-ce un ensemble de constantes ?



```
class NomType {  
    type    champs1;  
    type    champs2;  
    type[]  champs3;  
    ...  
}
```

```
enum NomType {  
    VALEUR1, VALEUR2,  
    ..., VALEURN;  
}
```

A sauvegarder dans un fichier nommé : NomType.java

Créer ses propres types

- Créer les fonctions suivantes
 - **NomType newNomType (...)** : pour encapsuler la création et l'initialisation d'une nouvelle valeur de votre type
 - Il est possible d'avoir plusieurs `newNomType` pour peu que les paramètres soient différents (**surcharge**)
 - **String toString(NomType variable)** : pour représenter sous forme d'une chaîne les informations (ou une partie) contenues dans votre type
- Pour lire ou modifier le contenu d'un champs d'un type, on utilise la notation pointée
 - `variable.champs / variable.champs = ...`

