# ITC34 - Algorithmique et programmation

## Partie C / C++

Benoît DARTIES

Benoit.Darties@u-bourgogne.fr

# Généricité paramétrique

- Intérêt de la généricité paramétrique :
  - Possibilité d'énoncer des descriptions de fonctions
  - Certains éléments restent formels
  - Eléments vus comme des paramètres
  - Indéfinis dans la définition de la fonction
  - Par la suite, correspondance à :
    - un type
    - un objet
    - une fonction
    - ou  une valeur

- Vocabulaire associé
  - Modèle de fonctions
  - Template en dialecte C++
  - polymorphisme paramétrique

# Généricité paramétrique

- Fonction affichant un tableau d'entiers

```
void afficheTab(int T[], int taille) {
  int i;
  int elementCourant;
  for (i = 0; i < taille ; i++ ) {
    elementCourant = T[i];
    cout <<  elementCourant  << ' ' ;
  }
}
```

- Fonction affichant un tableau de caractères ?
- Fonction affichant un tableau de flottants ?
- Fonction affichant un tableau de booléens ?
- ..

# Généricité paramétrique

- Création d'un paramètre formel identifiant un type de variables
  - Déclaration d'un type non défini «a priori»
  - Besoin d'un identificateur
  - mot-clé **`template <...>`**
  - portée limitée au bloc immédiatement consécutif à la déclaration

```
template< typename monTypeFormel >
```

déclaration d'un paramètre formel : mot-clé template

ce paramètre formel sera un type : mot-clé typename

Identificateur du type formel. ici : monTypeFormel

# Généricité paramétrique

- Création d'un paramètre formel identifiant un type de variables
  - Déclaration d'un type non défini «a priori»
  - Besoin d'un identificateur
  - mot-clé **`template <...>`**
  - portée limitée au bloc immédiatement consécutif à la déclaration

```
template< typename monTypeFormel >

void afficheTab(monTypeFormel T[], int taille) {
  int i;
  monTypeFormel elementCourant;
  for (i = 0; i < taille ; i++ ) {
    elementCourant = T[i];
    cout <<  elementCourant  << ' ' ;
  }
  cout << endl;
}
```

Utilisation du type formel , ajout d'un niveau d'abstraction supplémentaire

# Généricité paramétrique

- Utilisation d'un modèle de fonctions
  - Modèle précédemment créé
  - Besoin d'informer le compilateur du(des) modèle(s) à utiliser
  - Lors de l'appel
    - annonce du paramètre réel
    - correspond au paramètre formel de la déclaration

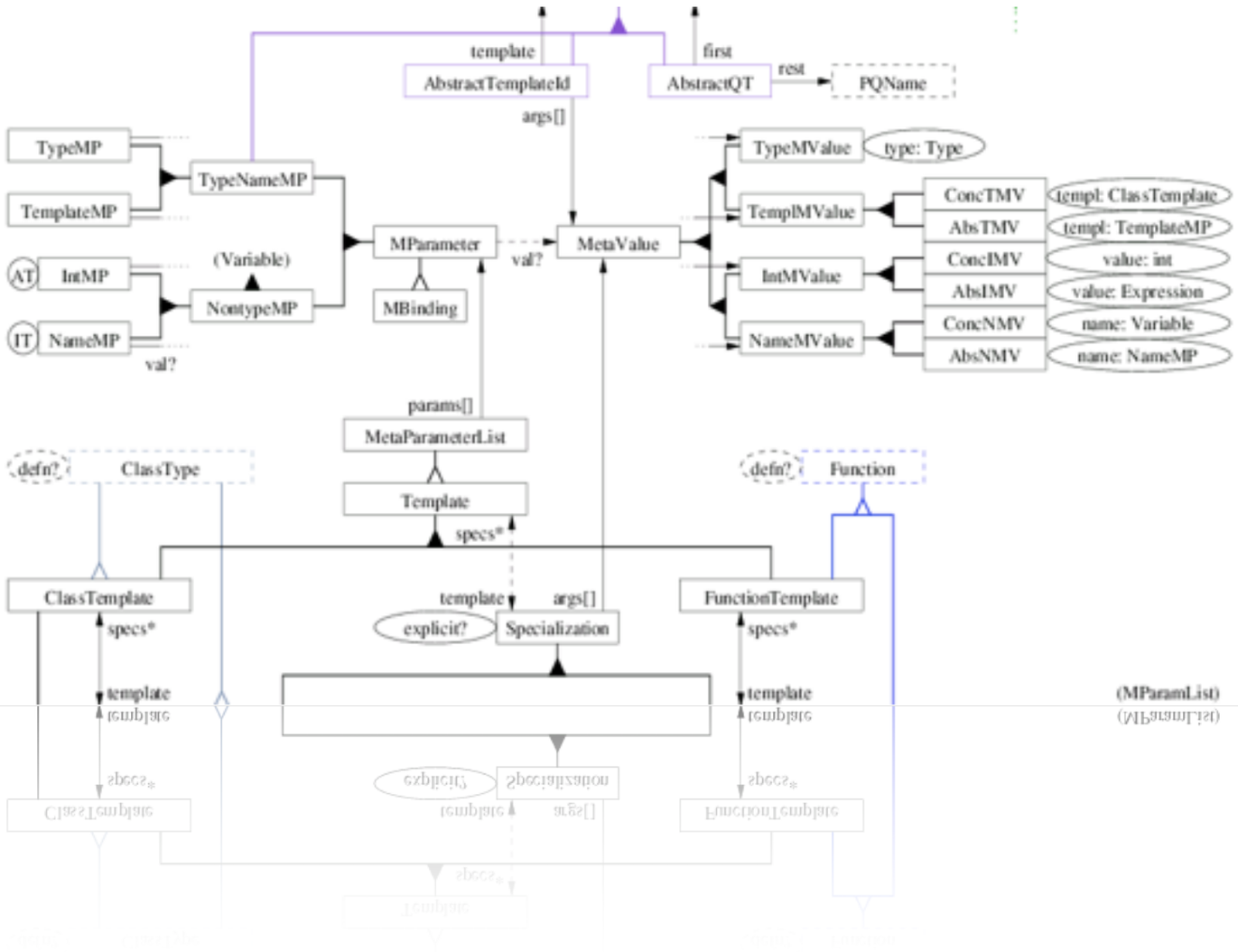
```
int main() {

  int tab[5] = {1, 2, 3, 4, 5};
  afficheTab<int>(tab, 5);

  float tab2[5] = {1.3, 2.2, 3.1, 4.4, 5.5};
  afficheTab<float>(tab, 5);
}
```

Précision de la valeur du paramètre formel

# Généricité paramétrique

- La bibliothèque STL
  - Standard Template Library
  - Librairies de containeurs, d'algorithmes, permettant de manipuler des listes et collections d'éléments
  - Association entre différents acteurs :
    - des structures avancées (ex. listes chaînées)
    - de la généricité paramétrique.
    - des méthodes de traitements

- Types de containeurs
  - Listes : list
  - Tableaux : vector
  - Listes triées : set
  - Listes d'associations : map




# Généricité paramétrique sur classes

# Généricité paramétrique sur classes

- Généricité paramétrique sur classes
  - Principe identique à la généricité paramétrique sur fonctions
  - Enoncer des descriptions de classe dans lesquelles certains éléments restent formels
  - Eléments vus comme paramètres
    - indéfinis au moment de la définition de classe
    - annoncés lors de leur utilisation
  - Template de classes / polymorphisme paramétrique

# Généricité : un exemple sur classe

# Généricité : un exemple sur classe


# Demo : exécution du scnéario

# Généricité : un exemple sur classe

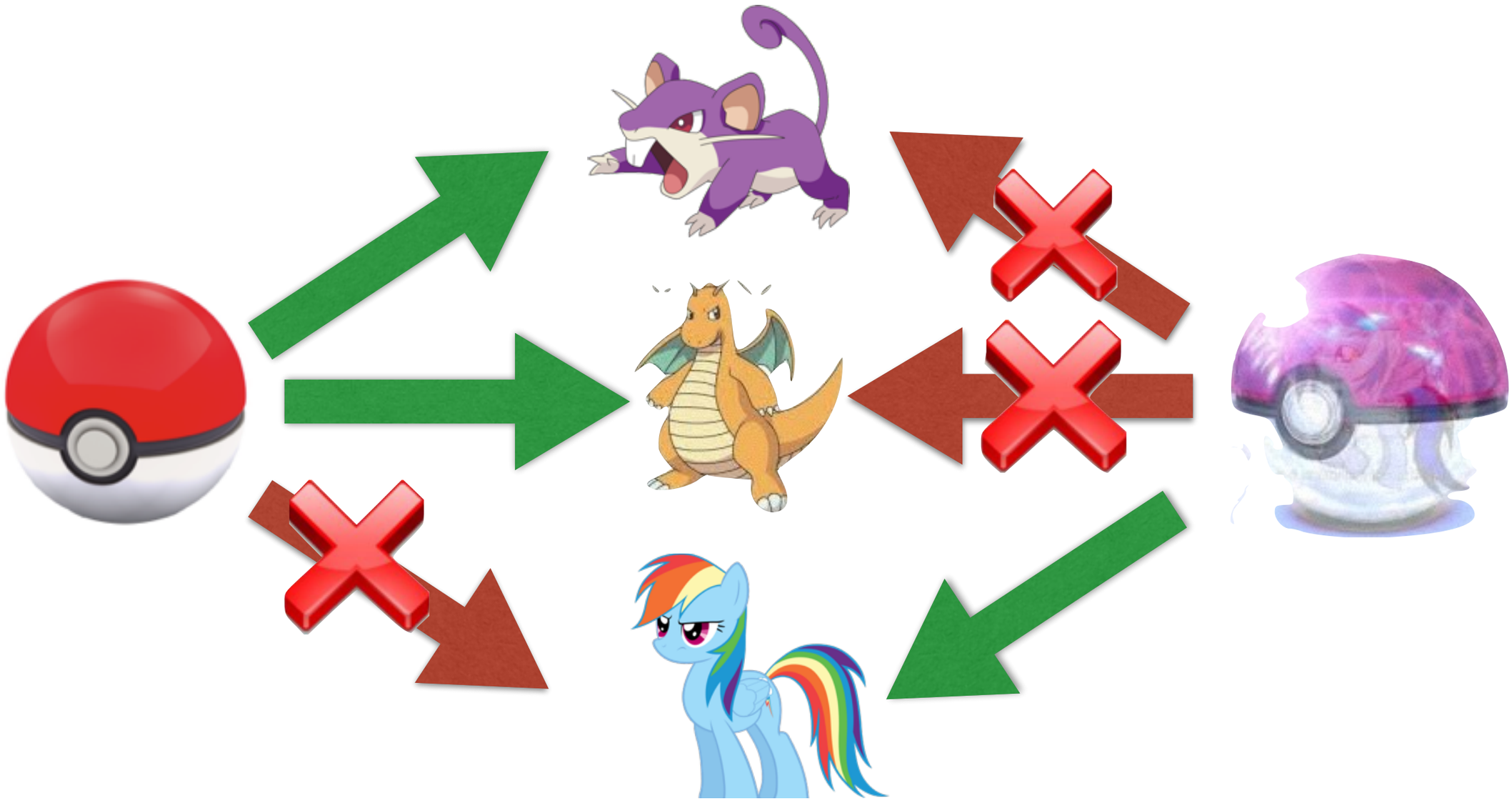
**ERREUR**

Pokeball = pour pokemon seulement!


# Demo : l'erreur en live

# Généricité : un exemple sur classe

## Pokeball = pour pokemon seulement!

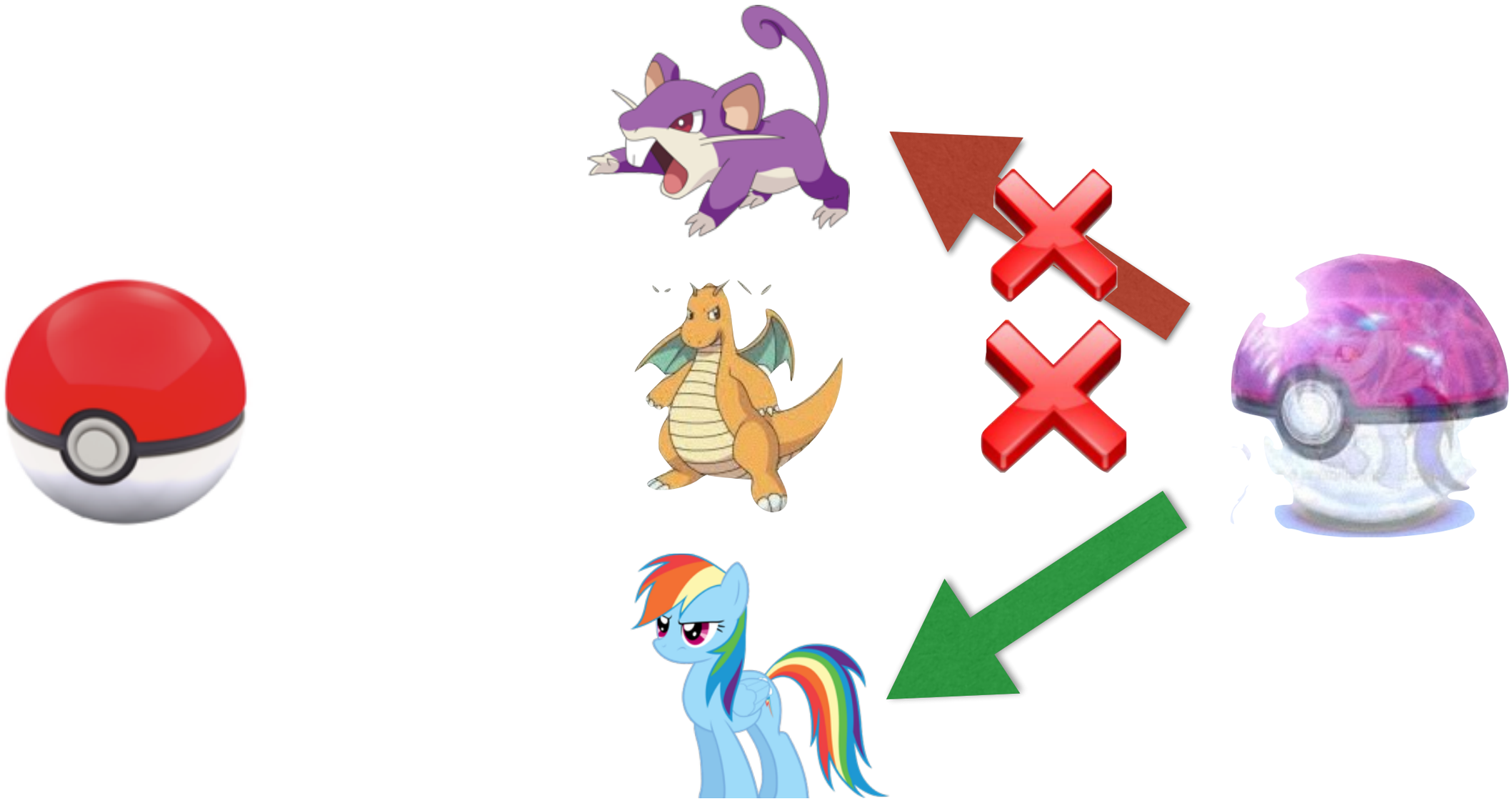


Pour capturer un Petit Poney : besoin de définir une pokéball spécifique pour les PP (une « Ponyball » )


# Demo : solution sans généricité = mauvaise solution

# Utilisation de la généricité

## Pokeball = pour pokemon seulement!



Pour capturer un Petit Poney : besoin de définir une pokéball spécifique pour les PP (une « Ponyball » )

# Généricité paramétrique sur classes

- Exemple : création de piles

| **Pile_int** |
| --- |
| **elements** : liste d'entiers |
| **Empiler(int) :** booleen<br>**Depiler()** : entier<br>**Vider()** : booleen<br>**nbElements()** : entier |

Pile d'entiers

| **Pile_float** |
| --- |
| **elements** : liste de flottants |
| **Empiler(float) :** booléen<br>**Depiler()** : flottant<br>**Vider()** : booleen<br>**nbElements()** : entier |

Pile de flottants

| **Pile_Vaisseau** |
| --- |
| **elements** : liste de vaisseaux |
| **Empiler(Vaisseau) :** booléen<br>**Depiler()** :Vaisseau<br>**Vider()** : booleen<br>**nbElements()** : entier |

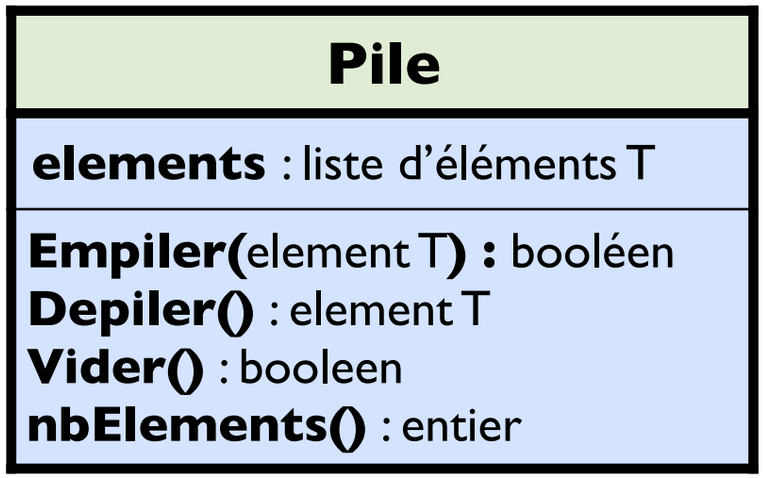
Pile de vaisseaux

Intérêt de la généricité :

- Créer une classe PIle dont le type d'éléments n'est pas défini
- Factorisation de code : I seule classe définie
- Définition du type d'éléments lors de l'utilisation seulement

# Généricité paramétrique sur classes

- Exemple : création de piles




Pile d'un type formel T

Création d'une pile = définition du paramètre formel : **Spécialisation**

- Pile<int>
- Pile<float>
- Pile<Vaisseau>
- ....

# Généricité paramétrique sur classes

- Définition d'une classe avec paramètres formels

```
template <typename typeElement>
class Pile {
  private :
    Vector<typeElement> elements;

  public :
    bool empiler(typeElement T) { ... }
    typeElement depiler() { ... }
    bool vider() { ... }
    int nbElements() { ... }
}
```

Utilisation d'un type formel comme type d'attribut

```
Pile<int> pileEntiers;
Pile<Vaisseau> pileVaisseaux;
```

# Généricité paramétrique sur classes

- Définition de méthodes génériques à l'extérieur de la classe :
  - Méthodes déportées
  - Besoin de rappeler les éléments génériques avant chaque définition

```
template <typename typeElement>
class Pile () { ... };

// définition hors classe
template <typename typeElement>
bool Pile::empiler(typeElement T) { ... }

template <typename typeElement>
typeElement Pile::depiler() { ... }

bool vider() { ... }
int nbElements() { ... }
```

# Les conteneurs

# Vector etc