M3103 – Algorithmique avancée

TP 2.1 *(associé au Cour 2)*

Arbres Binaires de Recherche – implantation du TD 2.1

(2 séances encadrées)

Pour ce TP

- créer un projet **M3103 – ArbreNoeudBinaireRecherche** dans NetBeans (cf. Aide NetBeans).
- copier les fichiers à compléter disponibles dans /users/info/pub/2a/M3103/tp21 dans le répertoire créé par NetBeans (cf. Aide NetBeans).
- importer les fichiers copiés (cf. Aide NetBeans).
- créer la configuration **TesteABR** (cf. Aide NetBeans).
- Préparer la configuration **TesteArbre** avec comme point d'entrée **`testeArbre.cpp`**.

Excercie 1 : Les trois méthodes d'affichage

Dans le fichier **`ArbreNoeudBinaireRecherche.cpp`** implanter les workers récursifs des trois méthodes d'affichage des informations portées par les nœuds d'un ABR :

- **`affichePrefixe()`**
- **`afficheInfixe()`**
- **`affichePostfixe()`**

Excercie 2 : Insertion associative dans un ABR (récursif)

On fournit la méthode publique suivante de la classe **ArbreNoeudBinaireRecherche** :

```
void ArbreNoeudBinaireRecherche<TypeInfo>::insere(const TypeInfo& nouvelleInfo) {
    insertWorker(ptrRacine, nouvelleInfo);
}
```

Implanter le worker privé récursif qui fait le travail.

```
void ArbreNoeudBinaireRecherche<TypeInfo>::insertWorker(
                        NoeudBinaire<TypeInfo>*& ptrRac, const TypeInfo& nouvelleInfo);
```

Tester les workers des exercices 1 et 2 avec la procédure **`testeInsertAffiche()`** dans **`testeArbreNoeudBinaireRecherche.cpp`**.

Excercie 3 : Hauteur d'un arbre binaire (récursif)

On fournit la méthode publique suivante de la classe **ArbreNoeudBinaireRecherche** :

```
int ArbreNoeudBinaireRecherche<TypeInfo>::getHauteur() const {
    return getHauteurWorker(ptrRacine);
}
```

Implanter l'algorithme du worker privé récursif qui fait le travail.

```
int ArbreNoeudBinaireRecherche<TypeInfo>::getHauteurWorker(
                                                NoeudBinaire<TypeInfo>* ptrRac) const;
```

Tester ce worker avec la procédure **`testeHauteur();`** dans **`testeArbreNoeudBinaireRecherche.cpp`**.

Excercie 4 : Nombre de nœuds d'un arbre binaire (récursif)

On fournit la méthode publique suivante de la classe **ArbreNoeudBinaireRecherche** :

```
/**
  * @return  nombre de noeuds de cet arbre
  */
int ArbreNoeudBinaireRecherche<TypeInfo>::getNombreDeNoeuds() const {
    return getNombreDeNoeudsWorker(ptrRacine);
}
```

Implanter l'algorithme du worker privé récursif qui fait le travail.

```
int ArbreNoeudBinaireRecherche<TypeInfo>::getNombreDeNoeudsWorker(
                                              NoeudBinaire<TypeInfo>* ptrRac) const;
```

Tester ce worker avec la procédure **`testeGetNombreNoeuds();`** dans **`testeArbreNoeudBinaireRecherche.cpp`**.

Excercie 5 : Nombre de feuilles d'un arbre binaire (récursif)

On fournit la méthode publique suivante de la classe **ArbreNoeudBinaireRecherche** :

```
/**
  * @return  nombre de feuilles de cet arbre
  */
int ArbreNoeudBinaireRecherche<TypeInfo>::getNombreDeFeuilles() const {
    return getNombreDeNoeudsWorker(ptrRacine);
}
```

Implanter l'algorithme du worker privé récursif qui fait le travail.

```
int ArbreNoeudBinaireRecherche<TypeInfo>::getNombreDeFeuillesWorker(
                                              NoeudBinaire<TypeInfo>* ptrRac) const;
```

Tester ce worker avec la procédure **`testeGetNombreFeuilles();`** dans **`testeArbreNoeudBinaireRecherche.cpp`**.

Excercie 6 : Recherche associative dans un ABR (récursif)

On fournit la méthode publique suivante de la classe **ArbreNoeudBinaireRecherche** :

```
/**
  * @return  true si unInfo est présente dans cet arbre ; false sinon
  */
bool ArbreNoeudBinaireRecherche<TypeInfo>::estInfoPresente(
                                                  const TypeInfo& uneInfo) const {
    return estInfoPresenteWorker(ptrRacine, uneInfo); // nullptr is same as false
}
```

Implanter le worker privé récursif vu en cours qui fait le travail.

```
bool ArbreNoeudBinaireRecherche<TypeInfo>::estInfoPresenteWorker(
                    NoeudBinaire<TypeInfo>* ptrRac, const TypeInfo& infoCible) const;
```

Tester ce worker avec la procédure **`testeEstInfoPresente();`** dans **`testeArbreNoeudBinaireRecherche.cpp`**.

Excercie 7 : Plus grande valeur dans un ABR (récursif)

On fournit la méthode publique suivante de la classe **ArbreNoeudBinaireRecherche** :

```
TypeInfo ArbreNoeudBinaireRecherche<TypeInfo>::getMax() const throw (PrecondViolatedExcep);
```

Implanter l'algorithme du worker récursif qui fait le travail.

```
TypeInfo ArbreNoeudBinaireRecherche<TypeInfo>::getMaxWorker(
                                    const NoeudBinaire<TypeInfo>* ptrRac) const;
```

Tester ce worker avec la procédure **`testeGetMax();`** dans **`testeArbreNoeudBinaireRecherche.cpp`**.

Excercie 8 : Nombre d'occurrences d'une valeur (récursif)

On fournit la méthode publique suivante de la classe **ArbreNoeudBinaireRecherche** :

```
/**
  * @return  nombre d'occurrences de uneInfo dans cet arbre binaire de recherche
  */
int ArbreNoeudBinaireRecherche<TypeInfo>::getNombreOccurrences(
                                                    const TypeInfo& uneInfo) const ;
```

Implanter l'algorithme du worker privé récursif qui fait le travail.

```
int ArbreNoeudBinaireRecherche<TypeInfo>::getNombreOccurrencesWorker(
                const NoeudBinaire<TypeInfo>* ptrRac, const TypeInfo& uneInfo) const;
```

Tester ce worker avec la procédure **`testeGetNombreOccurrences();`** dans **`testeArbreNoeudBinaireRecherche.cpp`**.

Excercie 9 : Deux arbres ont-il la même géométrie (récursif)

On fournit la méthode publique suivante de la classe **ArbreNoeudBinaireRecherche** :

```
/**
  * @return  true si cet arbre et autreArbre ont le même dessin (sans les informations)
  */
bool ArbreNoeudBinaireRecherche<TypeInfo>::aMemeGeometrieQue(
                      const ArbreNoeudBinaireRecherche<TypeInfo>& autreArbre) const ;
```

Implanter l'algorithme du worker récursif qui fait le travail.

```
bool ArbreNoeudBinaireRecherche<TypeInfo>::aMemeGeometrieQueWorker(
                const NoeudBinaire<TypeInfo>* monPtrRac,
                const NoeudBinaire<TypeInfo>* sonPtrRac) const;
```

Tester ce worker avec la procédure **`testeAMemeGeometrieQue();`** dans **`testeArbreNoeudBinaireRecherche.cpp`**.

Annexe

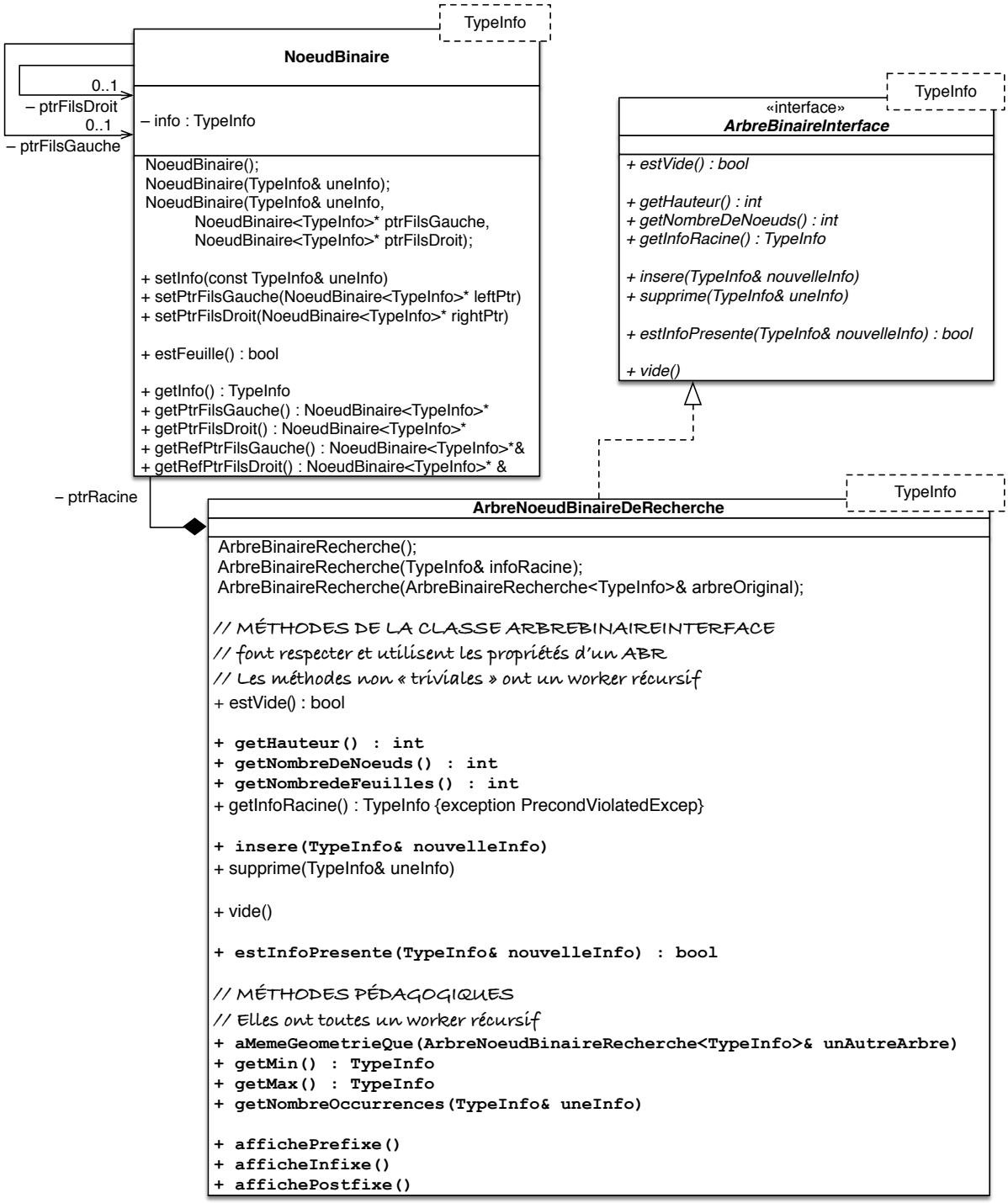

Figure 1 : Spécification UML des classes utilisées pour la classe ArbreNoeudBinaireRecherche