Fichier `Noeud.java`

```
// Definition de pool valide pour toutes les methodes.
static ExecutorService pool = Executors.newCachedThreadPool();


Integer psomme() {
  Integer[] resultats = new Integer[nbEnfants];
  Thread[] threads = new Thread[nbEnfants];

  for( int i = 0; i < nbEnfants; i++ ) {
    final int fi = i;
    threads[i] = new Thread( () -> resultats[fi] = enfants[fi].psomme() );
    threads[i].start();
  }

  Integer total = 0;
  for( int i = 0; i < nbEnfants; i++ ) {
    try {
      threads[i].join();
    } catch( InterruptedException e ) {
      assert false : "*** Erreur dans psomme";
    }
    total += resultats[i];
  }

  return valeur() + total;
}

//
```

```
@SuppressWarnings("unchecked")
Integer preduce( BinaryOperator<Integer> binop ) {
  Future<Integer>[] futures = new Future[nbEnfants];
  for( int i = 0; i < nbEnfants; i++ ) {
    final int fi = i;
    futures[i] = pool.submit( () -> enfants[fi].preduce(binop) );
  }

  Integer total = valeur();
  try {
    for( int i = 0; i < nbEnfants; i++ ) {
      total = binop.apply( total, futures[i].get() );
    }
  } catch( ExecutionException | InterruptedException e ) {
    assert false : "*** Erreur dans preduce";
  }

  return total;
}

//
```

```
@SuppressWarnings("unchecked")
Arbre pmap( UnaryOperator<Integer> f ) {
  Future<Arbre>[] futures = new Future[nbEnfants];
  for( int i = 0; i < nbEnfants; i++ ) {
    final int fi = i;
    futures[i] = pool.submit( () -> enfants[fi].pmap(f) );
  }

  Integer val = f.apply( valeur() );

  Arbre[] nouveauxEnfants = new Arbre[nbEnfants];
  for( int i = 0; i < nbEnfants; i++ ) {
    try {
      nouveauxEnfants[i] = futures[i].get();
    } catch( ExecutionException | InterruptedException e ) {
      assert false : "*** Erreur dans pmap";
    }
  }

  return new Noeud( val, nouveauxEnfants );
}
```