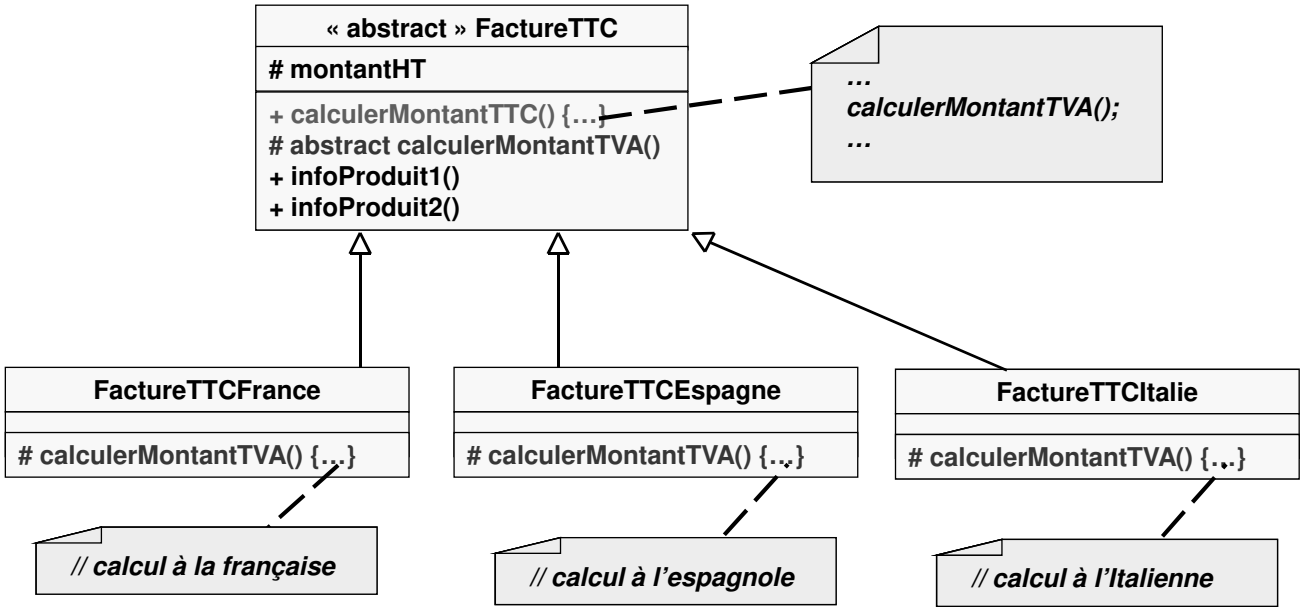
# *Un petit problème de conception…*

- … d'une application de facturation à l'échelle de l'Union Européenne
  - On souhaite créer et manipuler des objets « factures » avec
    - Un montant HT
    - Différentes méthodes donnant des informations diverses sur le produit facturé
    - Une méthode calculerMontantTTC() de calcul du montant TTC
  - Contexte (métier)
    - Les taux de TVA dépendent du pays où l'achat est effectué, ainsi que les règles de calcul du montant de la TVA (exonérations, taux différenciés…) qui sont (supposées) complexes.
    - On aura donc des factures « françaises », « italiennes », « espagnoles », etc.
  - Hypothèse (de travail, pour l'exercice !) : le calcul du montant TTC est une opération complexe (!)



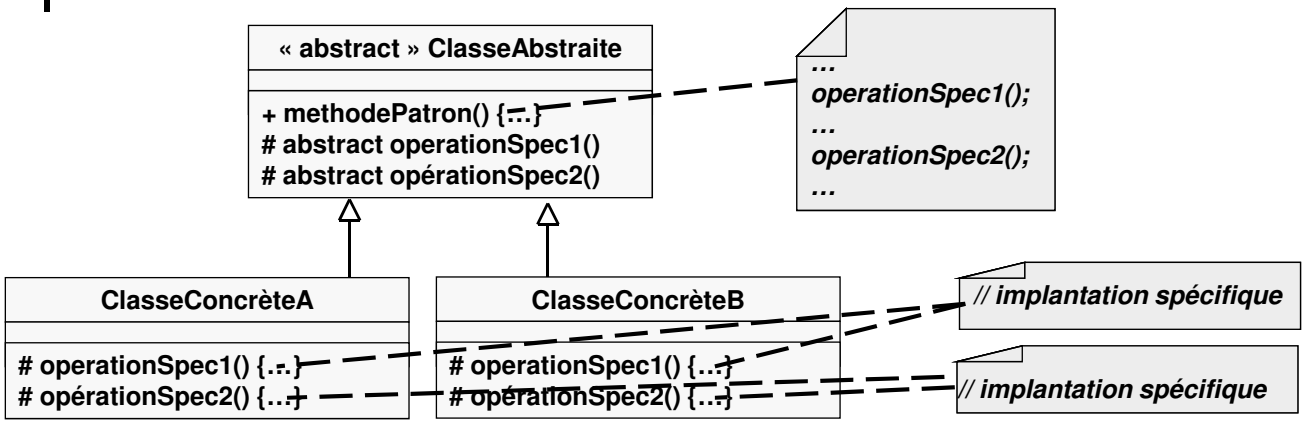


# *Un petit problème de conception…*




- Cette solution est connue sous le nom de « template méthod » ou « patron de méthode »

# *Le design pattern « patron de méthode »*

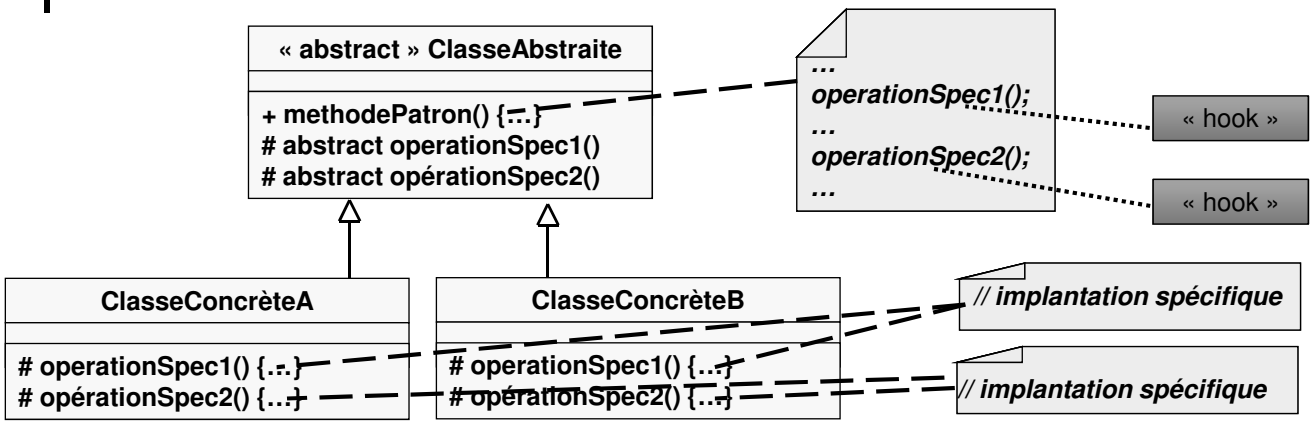



- Objectif
  - Partager du code commun
  - Reporter dans les sous-classes une partie d'une opération sur un objet
    - Algorithme (méthode) avec partie invariable et partie spécifique (variable)
- L'héritage supporte
  - La mise en facteur du code des parties communes (à des méthodes)
  - La spécialisation des parties qui diffèrent



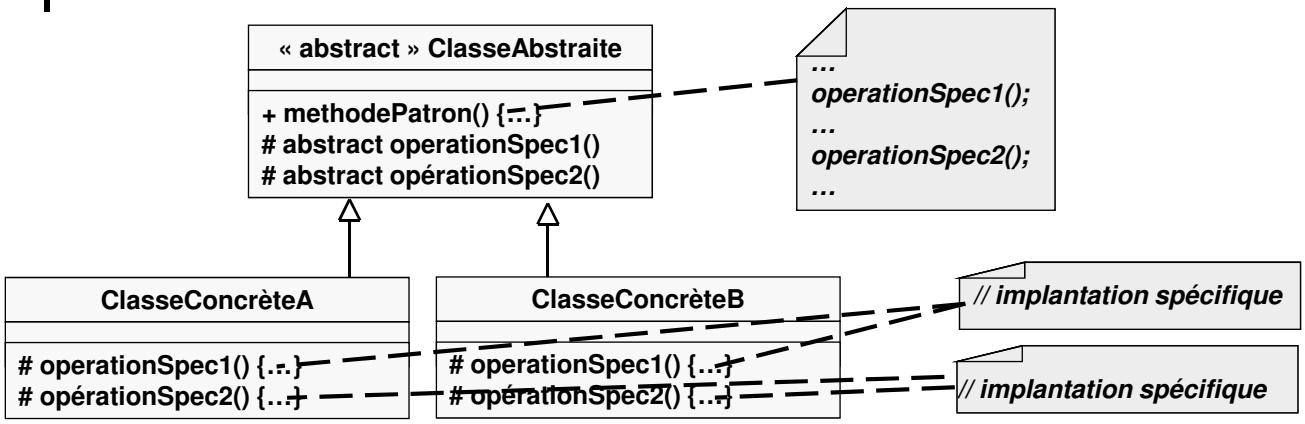


# *Le design pattern « patron de méthode »*




- methodePatron() : méthode générique
  - Partie commune
  - Fournit la structure générale de l'algorithme
  - Une partie de son implantation est définie dans la sous-classe (externalisation)
- operationSpec1() et operationSpec2() : parties spécifiques
  - Intégration *via* des « hooks » dans methodePatron()

# *Le design pattern « patron de méthode »*




- Intention (objectif) : organisation du code
  - Patron de méthode est un pattern « de niveau classe » (*vs* « de niveau objet »)
- S'occupe du comportement de l'entité
  - Rôle « comportemental » (*vs* « structurel » ou « créationnel »)
- Utilise le polymorphisme