Travaux Dirigés No3
Conception par Contrats

Frédéric Peschanski

2 février 2016

Dans ce TD nous nous mettons en pratique la conception par contrat à partir des spécifications semi-formelles élaborées lors du TD précédent.

Exercice 1 : Compte bancaire

Décrire une interface Java Compte pour le service de compte bancaire du TD2 (cf. annexes).

Indiquer également les invariants et les préconditions/postconditions sur les méthodes (contrats).

Décrire une implémentation des contrats séparant le code métier (CompteImpl) du code de contractualisation (CompteContract). Indiquer le *design pattern* employé.

Exercice 2 : Agence bancaire

Mêmes questions pour le service Agence

Annexe A : Spécifications du compte

service : Compte
observators :
  const nom : [Compte] → String
  const numero : [Compte] → int
  solde : [Compte] → double
  const limite : [Compte] → double
  montantDecouvert : [Compte] → double
    pre montantDecouvert(C) **require** estDecouvert(C)
  estDecouvert : [Compte] → boolean
  peutPrelever : [Compte] × double → boolean
    pre peutPrelever(C,s) **require** $s > 0$
Constructors :
  init : String × int × double → [Compte]
    pre init(n,num,dec) **require** $n \neq$ "" $\wedge\ num > 0 \wedge dec \geq 0$
  init : Compte → [Compte]
Operators :
  depot : [Compte] × double → [Compte]
    pre depot(C,s) **require** $s > 0$

  retrait : [Compte] × double → [Compte]
    pre depot(C,s) **require** peutPrelever(C,s)
Observations :

[invariants]
montantDecouvert(C) $\stackrel{\text{min}}{=}$ -solde(C)
estDecouvert(C) $\stackrel{\text{min}}{=}$ solde(C) < 0
peutPrelever(C,s) $\stackrel{\text{min}}{=}$ solde(C)-s $\geq$ limite(C)
solde(C) $\geq$ limite(C) *// invariant utile*
[init]
nom(init(n,num,dec)) = n
numero(init(n,num,dec)) = num
solde(init(n,num,dec)) = 0
limite(init(n,num,dec)) = - dec
nom(init(C)) = nom(C)
numero(init(C)) = numero(C)
solde(init(C)) = solde(C)
limite(init(C)) = limite(C)
[depot]
solde(depot(C,s)) = solde(C) + s
[retrait]
solde(retrait(C,s)) = solde(C) - s

Annexe B : Spécifications de l'agence

service : Agence
observators :
const nom : [Agence] → String
numeros : [Agence] → Set<int>
nbComptes : [Agence] → int
compteExiste : [Agence] × int → bool
getCompte : [Agence] × int → Compte
pre getCompte(A,num) **require** compteExiste(A,num)
Constructors :
init : String → [Agence]
pre init(n) **require** $n \neq$ ""
Operators :
creerCompte : [Agence] × String × int × double → [Agence]
pre creerCompte(A,n,num,dec) **require** ¬compteExiste(A,num)
fermerCompte : [Agence] × int → [Agence]
pre fermerCompte(A,num) **require** compteExiste(A,num)
virement : [Agence] × int × int × double → [Agence]
pre virement(A,n1,n2,s) **require** n1≠ n2 ∧ compteExiste(A,n1) ∧ compteExiste(A,n2)
∧ Compte::peutPrelever(getCompte(A,n1),s)
Observations :
[invariants]
nbComptes(A) $\stackrel{\text{min}}{=}$ card(numeros(A))
compteExiste(A,num) $\stackrel{\text{min}}{=}$ num $\in$ numeros(A)
[init]
nom(init(n)) = n
numeros(init(n)) = $\emptyset$
[creerCompte]
numeros(creerCompte(A,n,num,dec)) = numeros(A) ∪{ num }
getCompte(creerCompte(A,n,num,dec),num) = Compte::init(n,num,dec)
pour tout $n \in$ numeros(A), getCompte(creerCompte(A,n,num,dec),n) = getCompte(A)
[fermerCompte]
numeros(fermerCompte(A,num)) = numeros(A) \{num}
pour tout $n \in$ numeros(A)\{num}, getCompte(fermerCompte(A,num,dec),n) = getCompte(A)
[virement]
numeros(virement(A,n1,n2,s)) = numeros(A)
getCompte(virement(A,n1,n2,s),n1) = Compte::retrait(getCompte(A,n1),s)
getCompte(virement(A,n1,n2,s),n2) = Compte::depot(getCompte(A,n2),s)
$\forall n \in$ numeros(A) \{n1,n2}, getCompte(virement(A,n1,n2,s),n) = getCompte(A)