Travaux sur Machine Encadrés No3
Conception par Contrats

Frédéric Peschanski

10 février 2014

Exercice : Projet files à priorité

On souhaite dans cet exercice traduire en **conception par contrat** la spécification de files à priorité proposée en annexe (page suivante).

Pour cela on souhaite implémenter :
- une interface java FilesPrio<T> et le contrat (en commentaires) correspondant à la spécification proposée,
- une implementation du contrat FilesPrioContract<T> dans le cadre du *design pattern* décorateur,
- deux implémentations différentes FilesPrioImpl<T> et FilesPrioImplBug<T>, la seconde étant une version erronée de la première (soyez imaginatifs pour les bugs), ainsi que
- des exemples d'utilisation dans FilesPrioMain, avec ou sans vérification en ligne du contrat.

Remarque 1 : le contrat décrit dans l'interface et implémenté ensuite devra respecter au mieux les spécifications.

Remarque 2 : on pourra dans les annotations sémantiques (commentaires dans l'interface) utiliser des quantificateurs \forall, \exists, etc. ainsi que des constructions ad-hoc que l'on expliquera proprement. L'implémentation du contrat devra être la plus systématique possible (même type d'annotation = même algorithme de vérification).

Remarque 3 : on pourra affiner ou modifier les spécifications si nécessaires

Le projet devra être construit avec un fichier de construction build.xml pour `ant` proposant :
- une cible clean pour nettoyer le projet,
- une cible compile pour compiler les sources,
- une cible run pour exécuter les exemples,
- et une cible dist pour générer l'archive.

Important : la soumission se fera dans un fichier jar gzippé portant le nom TME3-CPS-NOM1-NOM2.jar.gz. Ce jar devra être exploitable en dehors de tout environnement de développement.

Annexe : Spécifications

service : FilesPrio<T>
types : int, boolean, Set
observators :
size : [FilesPrio] → int
empty : [FilesPrio] → boolean
activePrios : [FilesPrio] → Set<int>
isActivePrio : [FilesPrio] × int → boolean
maxPrio : [FilesPrio] → int
sizePrio : [FilesPrio] × int → int
getPrio : [FilesPrio] × int → T
pre getPrio(P,i) **require** sizePrio(P,i)> 0
get : [FilesPrio] → T
pre get(P) **require** size(P)> 0
getElem : [FilesPrio] × int × int → T
pre getElem(P,i,k) **require** $i \in$ activePrios(P) $\wedge\ 0 < k \leq$sizePrio(P,i)
Constructors :
init : → [FilesPrio]
Operators :
putPrio : [FilesPrio] × int × T → [FilesPrio]
pre putPrio(P,i,e) **require** i$\geq 0 \wedge$ e$\neq$ null
put : [FilesPrio] × T → [FilesPrio]
pre put(P,e) **require** e$\neq$ null
removePrio : [FilesPrio] × int → [FilesPrio]
pre removePrio(P,i) **require** sizePrio(P,i)> 0
remove : [FilesPrio] → [FilesPrio]
pre remove(P) **require** size(P,i)> 0
Observations :
[invariants]
size(P) $\overset{\min}{=} \sum_{i \in \text{activePrios(P)}}$ sizePrio(P,i)
empty(P) $\overset{\min}{=}$ size(P)> 0
isActivePrio(P,i) $\overset{\min}{=}$ i $\in$ activePrios(P)
maxPrio(P) $\overset{\min}{=} max($activePrios(P)) avec $max(E) \overset{\text{def}}{=} x \in E \cup \{0\}$ t.q. $\forall y \in E,\ x \geq y$
getPrio(P,i) $\overset{\min}{=}$ getElem(P,i,1)
get(P) $\overset{\min}{=}$ getPrio(P,maxPrio(P))
$\forall i \in$ activePrios(P), sizePrio(P,i)> 0
$\forall i \notin$ activePrios(P), sizePrio(P,i)$= 0$
$\forall i \in$ activePrios(P), $\forall k \in [1..$sizePrio(P,i)$]$, getElem(P,i,k)$\neq$ null
[init]
size(init())$=0$
[putPrio]
isActivePrio(P,i) $\implies$ activePrios(putPrio(P,i,e)) = activePrios(P)
$\neg$isActivePrio(P,i) $\implies$ activePrios(putPrio(P,i,e)) = activePrios(P) $\cup\{i\}$
sizePrio(putPrio(P,i,e),i) = sizePrio(P,i) + 1
$\forall j \in$ activePrios(P)$\backslash\{i\}$, sizePrio(putPrio(P,i,e),j) = sizePrio(P,j)
getPrio(putPrio(P,i,e),i) = e
$\forall k \in [2..$sizePrio(P,i)+1$]$, getElem(putPrio(P,i,e),i,k) = getElem(P,i,k-1)
$\forall j \in$ activePrios(P)$\backslash\{i\}$, $\forall k \in [1..$sizePrio(P,j)$]$, getElem(putPrio(P,i,e),j,k) = getElem(P,j,k)
[put]
put(P,e) $\overset{\text{def}}{=}$ putPrio(P,e,maxPrio(P))
[removePrio]
sizePrio(P,i)>1 $\implies$ activePrios(removePrio(P,i)) = activePrios(P)
sizePrio(P,i)$=1$ $\implies$ activePrios(removePrio(P,i)) = activePrios(P) $\backslash\{i\}$
sizePrio(removePrio(P,i),i) = sizePrio(P,i) - 1
$\forall j \in$ activePrios(P)$\backslash\{i\}$, sizePrio(removePrio(P,i),j) = sizePrio(P,j)
$\forall k \in [1..$sizePrio(P,i)-1$]$, getElem(removePrio(P,i),i,k) = getElem(P,i,k)
$\forall i \in$ activePrios(P)$\backslash\{i\}$, $\forall k \in [1..$sizePrio(P,j)$]$, getElem(removePrio(P,i),j,k) = getElem(P,j,k)
[remove]
remove(P) $\overset{\text{def}}{=}$ removePrio(P,maxPrio(P))