Contract

[Champlain97]

Définition

- Nom: Contrat
- Auteur: Michel de Champlain

Résumé

- Le patron Contrat est vu comme un idiome qui permet l'utilisation d'assertions (pré et post-conditions), dans la définition d'une méthode

Objectif

- Fournir une implémentation de la conception par contrats, permettant le développement de classes fiables et robustes en Java

Motivation (1/2)

- Parfois, il est nécessaire de restreindre l'interface d'une classe de manière à réduire le domaine de données d'entrée

Motivation (2/2)

- Prenez par exemple un compteur borné, qui possède des bornes inférieures et supérieures:
  - La seule manière d'assurer la fiabilité du compteur est d'utiliser les exceptions. Celles-ci n'améliorent pas la validité de la classe et n'assurent pas le comportement de ses méthodes
  - Cette situation peut être améliorée par l'utilisation d'assertions définissant des contrats entre le compteur et ses clients.

Motivation (3/3)

- La méthode *inc()* déclare:
- Une pré-condition (require) qui demande que n soit positif et inférieur à la borne supérieure
- Une post-condition vérifiant que la méthode a été correctement codée

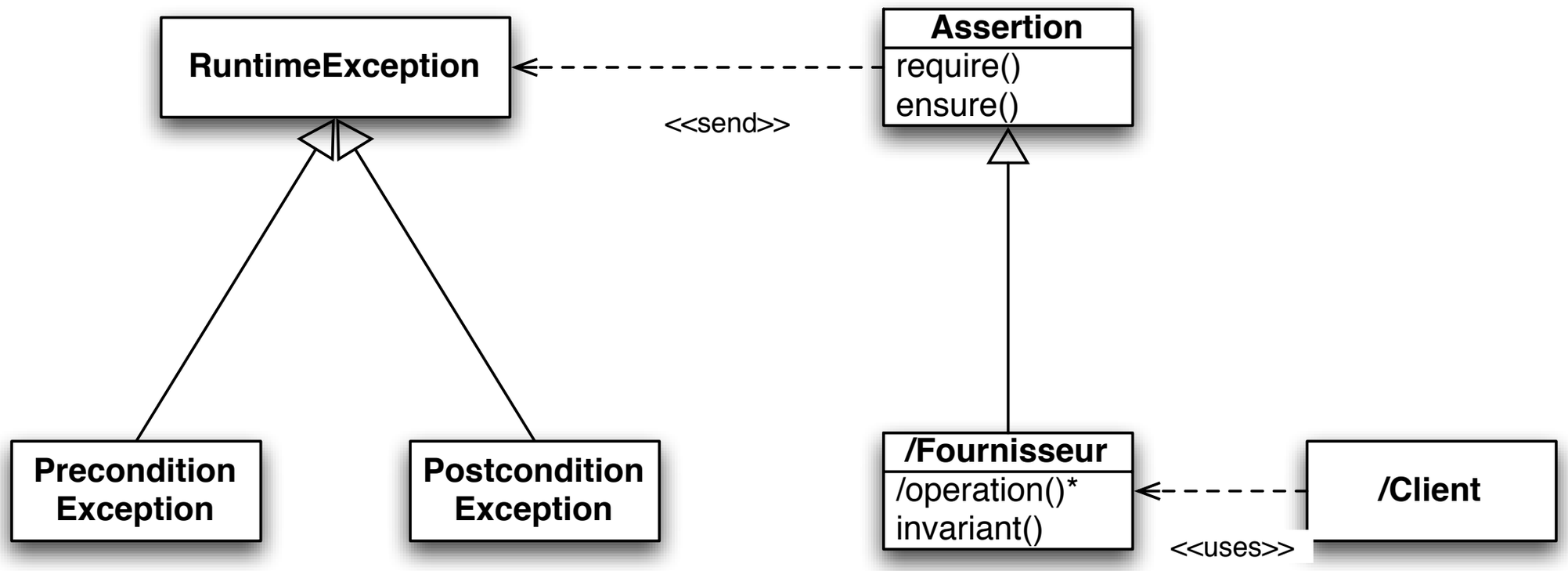
```
//  Compteur.java

public class Compteur {
    public void inc(int n) {
        require(n > 0 && count < upper);
        count += n;
        ensure(count == oldCount + n);
        ensure(invariant());
    }
    public boolean invariant() {
        return lower <= count && count <= upper;
    }
}
```

Applicabilité

- Etablir le comportement d'un objet
- Les assertions sont un élément de base pour une spécification plus formelle
- Spécification:
  - Définition d'une pré et d'une post-condition pour chaque méthode
  - Définition d'un invariant pour chaque classe

Structure


RuntimeException
Assertion
require()
ensure()
<<send>>
Precondition Exception
Postcondition Exception
/Fournisseur
/operation()*
invariant()
/Client
<<uses>>

Exemple de code

```
//  Assertion.java

public class Assertion {
    public static void require(boolean expr) {
        if (!expr) throw new PreconditionException();
    }
    public static void ensure(boolean expr) {
        if (!expr) throw new PostconditionException();
    }
}

class PreconditionException extends RuntimeException {
    PreconditionException() {
        super();
    }
}

class PostconditionException extends RuntimeException {
    PostconditionException() {
        super();
    }
}
```

Conséquences (1/2)

- Documentation des interfaces des classes
- Les assertions et les exceptions sont deux mécanismes complémentaires:
  - Les assertions spécifient des contrats
  - Les exceptions contrôlent les actions

Conséquences (2/2)

- Mieux que la programmation défensive
- Une extension simple pour plusieurs langages à objets qui:
  - utilisent les exceptions
  - combinent les assertions avec le contrôle d'exceptions

Conclusion

- Amélioration de la qualité d'un logiciel:
  - Validité
  - Robustesse
- Bien documenté en Eiffel, mais pas dans d'autres langages à objets
- Peut être simulé dans d'autres langages à objets: avec des exceptions et plusieurs extensions