

8. polymorphisme

GEF 321B
techniques de logiciel orienté objets
© greg phillips
collège militaire royal du canada
usage illimité permis dans la ministère de la défense nationale
autres usages sujet aux conditions stipulées
dans une licence publique gratuite communs
<http://creativecommons.org/licenses/by/2.0/ca/>

surcharge (overloading)

```
public class Stringifier {  
    public String convert(int x) {  
        return Integer.toString(x);  
    }  
    public String convert(double x) {  
        return Double.toString(x);  
    }  
    public String convert(char x) {  
        return Character.toString(x);  
    }  
    public static void main(String[] args) {  
        Stringifier s = new Stringifier();  
        s.convert(1);  
        s.convert(1.0);  
        s.convert('a');  
    }  
}
```

polymorphisme paramétrique

exemple Python

```
class Triangle:  
    def dessiner():  
        # dessiner triangle  
  
class Cercle:  
    def dessiner():  
        # dessiner cercle  
  
formes = [Triangle(), Cercle()]  
for forme in formes:  
    forme.dessiner()
```

polymorphisme paramétrique

exemple Java

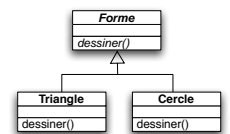
```
public class Triangle {  
    //...  
    public void dessiner() { /* dessiner triangle */ }  
}  
  
public class Cercle {  
    //...  
    public void dessiner() { /* dessiner cercle */ }  
}  
  
public static void main(String[] args) {  
    List formes = new ArrayList();  
    formes.add(new Triangle());  
    formes.add(new Cercle());  
    Iterator it = formes.iterator();  
    while (it.hasNext()) {  
        it.next().dessiner();  
    }  
}
```

classes abstraites et opérations abstraits

```
public abstract class Forme {  
    public abstract void dessiner();  
}
```

définition

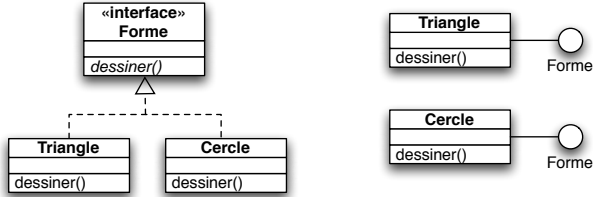
(classe abstrait)



```
public abstract class Forme {  
    public abstract void dessiner();  
}  
  
public class Triangle extends Forme {  
    //...  
    public void dessiner() { /* dessiner triangle */ }  
}  
  
public class Cercle extends Forme {  
    //...  
    public void dessiner() { /* dessiner cercle */ }  
}
```

interface

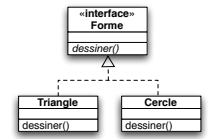
```
public interface Forme {  
    void dessiner();  
}
```



réalisation

définition

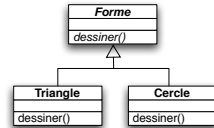
(interface)



```
public interface Forme {  
    void dessiner();  
}  
  
public class Triangle implements Forme {  
    //...  
    public void dessiner() { /* dessiner triangle */ }  
}  
  
public class Cercle implements Forme {  
    //...  
    public void dessiner() { /* dessiner cercle */ }  
}
```

usage

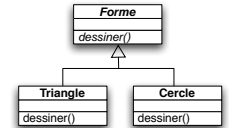
(Java 4)



```
public static void main(String[] args){  
    List formes = new ArrayList();  
    formes.add(new Triangle());  
    formes.add(new Cercle());  
    Iterator it = formes.iterator();  
    while (it.hasNext()) {  
        ((Forme)it.next()).dessiner();  
    }  
}
```

usage

(Java 5)



```
public static void main(String[] args){  
    List<Forme> formes = new ArrayList<Forme>();  
    formes.add(new Triangle());  
    formes.add(new Cercle());  
    for (Forme f : formes) {  
        f.dessiner();  
    }  
}
```

polymorphisme paramétrique

- méthodes avec le même nom et signature et **sens**, et la **liaison tardive** (late binding)
- dans un langage avec typage statique (e.g. Java), exige:
 - **supertype commune** qui déclare la méthode (classe ou interface)
 - **méthodes redéfinies** (overriden methods) dans les sous-classes

quand utiliser la polymorphisme?

pourquoi motifs de conception?

“ Chaque motif donne une description d’un problème qu’on trouve encore et encore dans votre environnement, et après il donne l’essence de la solution pour le problème, de telle façon que vous pouvez utiliser la solution un million de fois, sans répétition.”

– Christopher Alexander

“ Un motif de conception (design pattern) est une description d’une structure récurrente des composants communicants qui solutionne un problème général de conception dans un contexte particulier.”

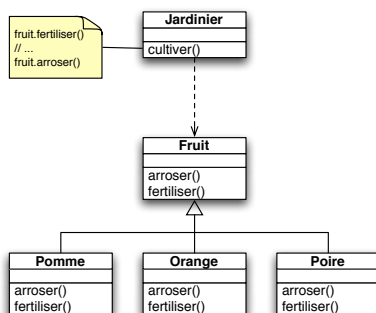
– Design Patterns. Gamma, Helm, Johnson, Vlissides, 1994

énoncé «case»

```
// arroser...
switch(typeDeFruit) {
case POMME:
    // arroser les pommes
    break;
case ORANGE:
    // arroser les oranges
    break;
case POIRE:
    // arroser les poires
    break;
}

// fertiliser...
switch(typeDeFruit) {
case POMME:
    // fertiliser les pommes
    break;
case ORANGE:
    // fertiliser les oranges
    break;
case POIRE:
    // fertiliser les poires
    break;
}
```

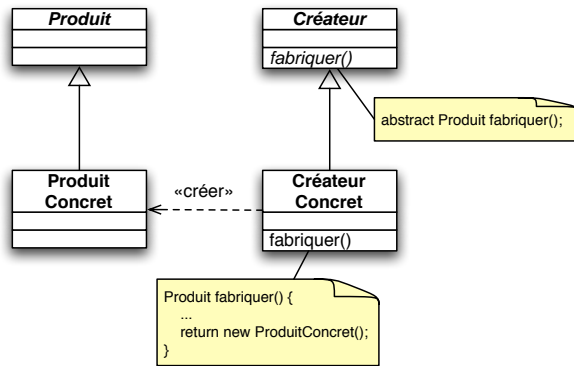
polymorphisme «meta-motif»



fabrication (factory method)

- superclasse définit interface pour la création d’un objet; c’est aux sous-classes à déterminer la classe créée
- souvent trouvé dans les boîtes à outils (toolkits) et les cadres d’applications (frameworks)

fabrication



références

- booch, rumbaugh & jacobson. uml language user guide. ch. 10 p. 125
- fowler & scott. uml distilled. 1997. ch. 4
- booch. object oriented analysis and design. ch. 3