

LIFI-Java 2004

Séance du Mercredi 29 sept.

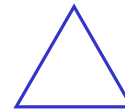
Cours 4

Classe abstraite (1/7)

- Problème:
 - Encapsuler des figures pour une `Tortue`
 - Paramétrer les figures

```
public class Tortue {  
    public void avance(double d);  
    public void tourne(double a);  
    public void leve();  
    public void baisse();  
};
```

```
Tortue t = new Tortue();  
  
for (int i=0;i<4;++i) {  
    t.avance(100);  
    t.tourne(90);  
}  
  
for (int i=0;i<3;++i) {  
    t.tourne(60);  
    t.avance(60);  
}
```



Classe abstraite (2/7)

- Une solution: créer des classes
 - Square, Triangle,...

```
public class Square {
    private double l;
    public Square(double l) { this.l=l; }
    public void applyTo(Tortue t) {
        for (int i=0;i<4;++i)
            t.avance(l);t.tourne(90)
    }
}

public class Triangle {
    private double l;
    public Triangle(double l) { this.l=l; }
    public void applyTo(Tortue t) {
        for (int i=0;i<4;++i)
            t.avance(l);t.tourne(90)
    }
}
```

```
Tortue t = new Tortue();

(new Square(100)).applyTo(t);
(new Square(200)).applyTo(t);

(new Triangle(60)).applyTo(t);
```

Classe abstraite (3/7)

- Une solution: créer des classes
- Ajouter des méthodes à Tortue

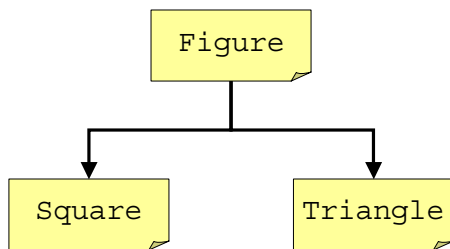
```
public class Tortue {  
    public void avance(double d);  
    public void tourne(double a);  
    public void leve();  
    public void baisse();  
    public void draw(Square s) {  
        s.applyTo(this);  
    }  
    public void draw(Triangle t) {  
        t.applyTo(this);  
    }  
};
```

```
Tortue t = new Tortue();  
  
t.draw(new Square(100));  
t.draw(new Square(200));  
  
t.draw(new Triangle(60));
```

Duplication de code!
Pas évolutif!

Classe abstraite (4/7)

- Faire dériver Square et Triangle d'une classe mère Figure



```
public class Figure {
    public void applyTo(Tortue t) { }
}
public class Square extends Figure {
    //...
    public void applyTo(Tortue t) {
        for (int i=0;i<4;++i)
            t.avance(1);t.tourne(90)
    }
}
public class Triangle extends Figure {
    //...
    public void applyTo(Tortue t) {
        for (int i=0;i<4;++i)
            t.avance(1);t.tourne(90)
    }
}
```

Classe abstraite (5/7)

- Factoriser la méthode `Tortue.draw()`

```
public class Tortue {  
    // ...  
    public void draw(Figure f) {  
        f.applyTo(this);  
    }  
};
```

```
public class Figure {  
    public void applyTo(Tortue t) { }  
}  
public class Square extends Figure {  
    //...  
    public void applyTo(Tortue t) {  
        for (int i=0;i<4;++i)  
            t.avance(1);t.tourne(90)  
    }  
}  
public class Triangle extends Figure {  
    //...  
    public void applyTo(Tortue t) {  
        for (int i=0;i<4;++i)  
            t.avance(1);t.tourne(90)  
    }  
}
```

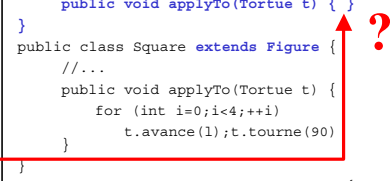
Classe abstraite (5/7)

- Factoriser la méthode `Tortue.draw()`

```
public class Tortue {  
    // ...  
    public void draw(Figure f) {  
        f.applyTo(this);  
    }  
};
```

- Question:
que doit faire la méthode `applyTo()` de la classe `Figure`?

```
public class Figure {  
    public void applyTo(Tortue t) { }  
}  
public class Square extends Figure {  
    //...  
    public void applyTo(Tortue t) {  
        for (int i=0;i<4;++i)  
            t.avance(1);t.tourne(90)  
    }  
}  
public class Triangle extends Figure {  
    //...  
    public void applyTo(Tortue t) {  
        for (int i=0;i<4;++i)  
            t.avance(1);t.tourne(90)  
    }  
}
```



Classe abstraite (6/7)

- Réponse:
 - rien!
 - c'est une classe *abstraite*!
- Java prévoit cela
 - déclarer la classe `abstract`
 - déclarer la méthode `abstract`
 - pas besoin de définir la fonction

```
public class Tortue {  
    // ...  
    public void draw(Figure f) {  
        f.applyTo(this);  
    }  
};
```

```
abstract public class Figure {  
    abstract public void applyTo(Tortue t);  
}  
public class Square extends Figure {  
    //...  
    public void applyTo(Tortue t) {  
        for (int i=0;i<4;++i)  
            t.avance(1);t.tourne(90)  
    }  
}  
public class Triangle extends Figure {  
    //...  
    public void applyTo(Tortue t) {  
        for (int i=0;i<4;++i)  
            t.avance(1);t.tourne(90)  
    }  
}
```


Classe abstraite (7/7)

- Une classe abstraite ne peut pas être instanciée
 - *Il faut dériver une classe pour définir l'implémentation de ses méthodes abstraites*
- Une classe abstraite peut aussi avoir des méthodes non abstraites
 - *qui utilisent des méthodes abstraites*

Encapsulation des types de base

- Le code suivant ne marche pas !?!

```
import java.util.Vector;

Vector v = new Vector();
v.addElement("salut"); // OK, "salut" est un String donc un Object

v.addElement(0);      // refusé à la compilation!
```

- 0 est un `int`, qui est un type de base
- un `vector` veut des objets!

↳ il faut encapsuler les types de base

Exemple: la classe Integer

- Encapsule le type `int`
- Pas équivalent à un `int`
 - appel *explicite* du constructeur
 - appel à `intValue()`
- Définit d'autres méthodes
 - static `int parseInt(String)`
 - boolean `compareTo(Integer)`
- Attention au test avec `==...`

```
import java.util.Vector;

Vector v = new Vector();

v.addElement(new Integer(0));
```

```
int a=Integer.parseInt("12");

Integer i=new Integer(3);
Integer j=new Integer(4);

// refusé à la compilation
if (i<j) {...}

// OK
if (i.compareTo(j)==-1) {...}
```

Références : le piège du test==

- == compare les références pas les valeurs!

```
Integer i=new Integer(3);
Integer j=new Integer(3);

if (i==j) {
    // on ne passera jamais ici!!!
}
```

```
Integer i=new Integer(3);
Integer j=new Integer(3);

if (i.compareTo(j)== 0) {
    // mais on passera là!!!
}
if (i.intValue() == j.intValue()) {
    // et là aussi!!!
}
```