
Cours 0 : Introduction

Jean-Michel Douin, douin au cnam point fr
version : 3 Septembre 2009

Notes de cours,

elles ne remplacent pas la lecture d'ouvrages ou de tutoriels sur ce sujet, cf. bibliographie

ESIEE

1

Sommaire

- **Présentation**

- java : Les objectifs des concepteurs,
- Présentation des concepts de l'orienté Objet,
- Conception à l'aide de patrons (*design pattern*).

- **Outils de développement**

- BlueJ
 - <http://www.bluej.org>
- JNEWS
 - <http://jfod.cnam.fr/jnews/>

ESIEE

2

Principale bibliographie utilisée pour ces notes

- [Grand00]
 - Patterns in Java le volume 1
<http://www.mindspring.com/~mgrand/>
- [head First]
 - Head first : <http://www.oreilly.com/catalog/hfdesignpat/#top>
- [DP05]
 - L'extension « Design Pattern » de BlueJ : <http://hamilton.bell.ac.uk/designpatterns/>
 - Ou bien en <http://www.patterncoder.org/>
- [Liskov]
 - Program Development in Java, Abstraction, Specification, and Object-Oriented Design, B.Liskov avec J. Guttag Addison Wesley 2000. ISBN 0-201-65768-6
- [divers]
 - Certains diagrammes UML : <http://www.dofactory.com/Patterns/PatternProxy.aspx>
 - informations générales <http://www.edlin.org/cs/patterns.html>

ESIEE

3

Java : les objectifs

- « Simple »
 - syntaxe " C »
- « sûr »
 - pas de pointeurs, vérification du code à l'exécution et des accès réseau et/ou fichiers
- Orienté Objet
 - (et seulement !), pas de variables ni de fonctions globales, types primitifs et objet
- Robuste
 - ramasse miettes, fortement typé, gestion des exceptions
- Indépendant d'une architecture
 - Portabilité assurée par la présence d'un interpréteur de bytecode sur chaque machine
- Environnement riche
 - Classes pour l'accès Internet
 - classes standard complètes
 - fonctions graphiques évoluées
- Technologie « Transversale »

ESIEE

4

Simple : syntaxe apparentée C,C++

```
public class Num{  
  
    public static int max( int x, int y){  
        int max = y;  
        if(x > y){  
            max = x;  
        }  
        return max;  
    }  
}
```

Note : C# apparenté Java

ESIEE

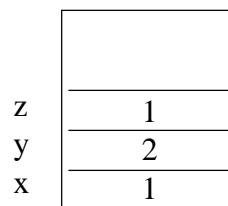
5

Sûr par l'absence de pointeurs (accessibles au programmeur)

- Deux types : primitif ou Object (et tous ses dérivés)

- primitif :

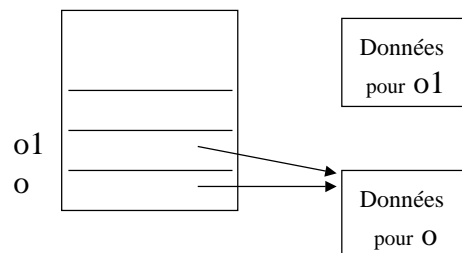
- int x = 1;
- int y = 2;
- int z = x;



- Object

- Object o = new Object();
- Object o1 = new Object();

- Object o1 = o;



ESIEE

6

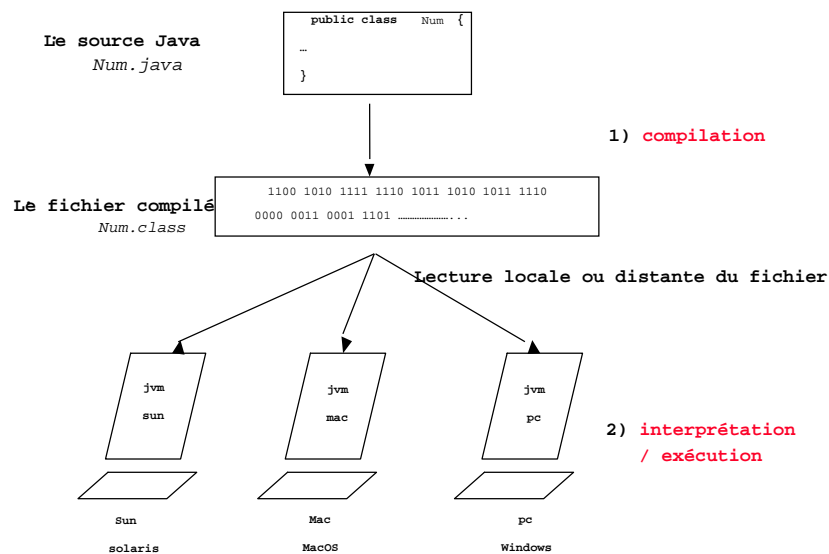
Robuste

- **Ramasse miettes ou gestionnaire de la mémoire**
 - Contrairement à l'allocation des objets, leur dé-allocation n'est pas à la charge du programmeur
 - (Ces dé-allocations interviennent selon la stratégie du gestionnaire)
- **Fortement typé**
 - Pas d'erreur à l'exécution due à une erreur de type
 - Mais un changement de type *hasardeux* est toujours possible
- **Généricité**
 - Vérification statique du bon « typage »
- **Exceptions**
 - Mécanisme de traitements des erreurs,
 - Une application ne devrait pas s'arrêter à la suite d'une erreur,
 - (ou toutes les erreurs possibles devraient être prises en compte ...)

ESIEE

7

Portable



ESIEE

8

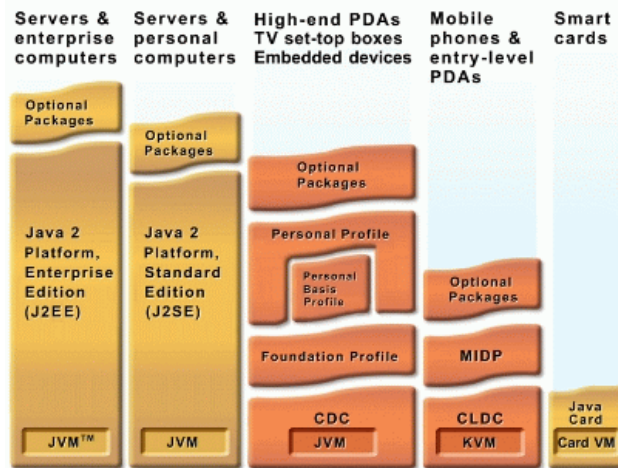
Environnement (très) riche

- java.applet
 - java.awt
 - java.beans
 - java.io
 - java.lang
 - java.math
 - java.net
 - java.rmi
 - java.security
 - java.sql
 - java.text
 - java.util
 - javax.accessibility
 - javax.swing
 - org.omg.CORBA
 - org.omg.CosNaming
- Liste des principaux paquetages de la plate-forme JDK 1.2
soit environ 1500 classes !!! Et bien d'autres A.P.I. JSJK, JINI, ...
- Le JDK1.3/1850 classes,
 - Le JDK1.5 ou j2SE5.0 3260 classes
 - Le J2SE 1.6

ESIEE

9

« Transversale »



Source : <http://java.sun.com/javame/technology/index.jsp>

ESIEE

10

Concepts de l'orienté objet

- Le vocable **Objet**, des mots
- Un historique ...
- Classe et objet (instance d'une classe)
- État d'un objet et données d'instance
- Comportement d'un objet et méthodes
 - liaison dynamique
- Héritage
- Polymorphisme

ESIEE

11

Un historique

- **Algorithm** + **Data Structures** = **Program**

A + **d** = **P** langage de type pascal, *années 70*

A + **D** = **P** langage modulaire, Ada, modula-2, *années 80*

a + **D** = **P** langage Orienté Objet *années 90, simula67...*

ESIEE

12

A + d = P

- **surface** (triangle t) =
- **surface** (carré c) =
- **surface** (polygone_régulier p) =
-
- **perimetre** (triangle t) =
- **perimetre** (carré c) =
- **perimetre** (polygone_régulier p) =
-
- *usage : import de la **librairie** de calcul puis*

```
carré unCarré; // une variable de type carré
```

```
y = surface ( unCarré)
```

ESIEE

13

A + D = P

- type carré = structure
- longueurDuCote
- fin_structure;
- >>>-----<<<
- **surface** (carré c) =
- **perimetre** (carré c) =
- (carré c) =
- *usage : import du **module carré** puis*

```
carré unCarré; // une variable de type carré
```

```
y = surface ( unCarré)
```

ESIEE

14

$$a + D = P$$

- classe **Carré** =
- longueurDuCote ...



- surface () =
- perimetre () =
- () =
- fin_classe;

- usage : import de la **classe carré** puis

– carré unCarré; // une instance de la classe Carré

- y = unCarré.surface ()

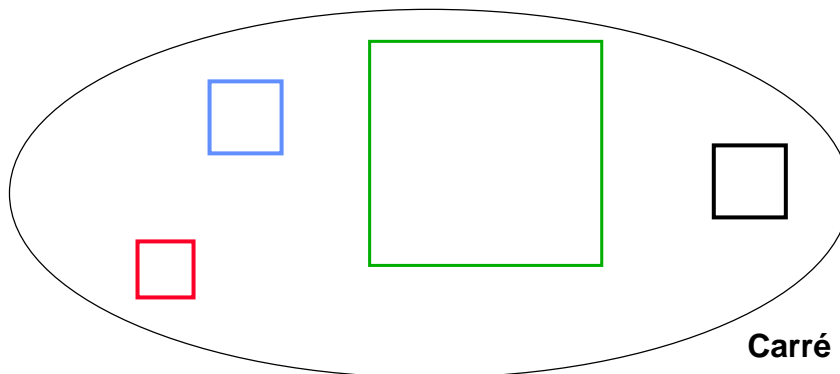
ESIEE

15

Classe et objet (instance d'une classe)

```
class Carré{
```

```
}
```

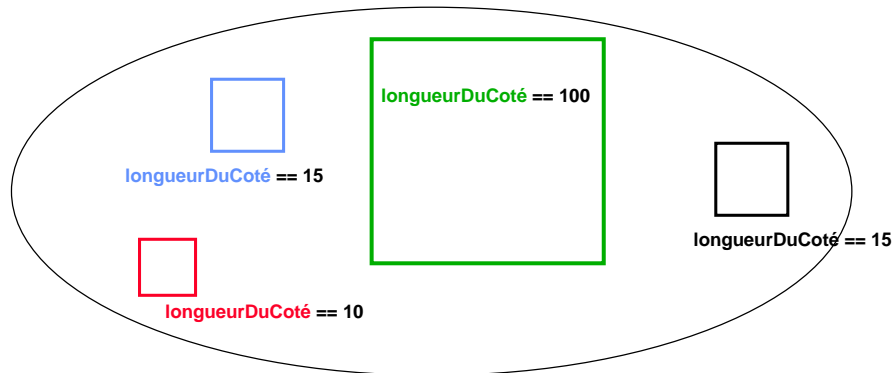


ESIEE

16

État d'un objet et données d'instance

```
class Carré{  
  int longueurDuCoté;  
  Couleur couleur;  
}
```

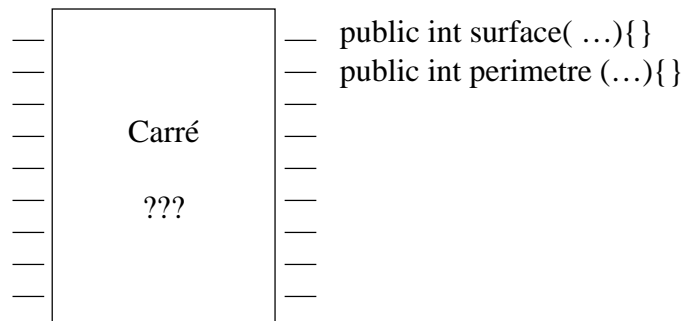


ESIEE

17

Classe et Encapsulation

- **contrat avec le client**
 - interface publique, une documentation, le nom des méthodes
 - implémentation privée
 - Les données d'instances déclarées private

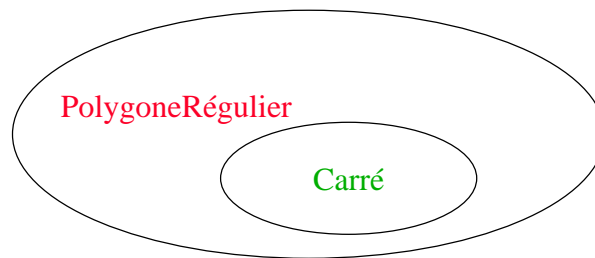


ESIEE

18

Héritage et classification

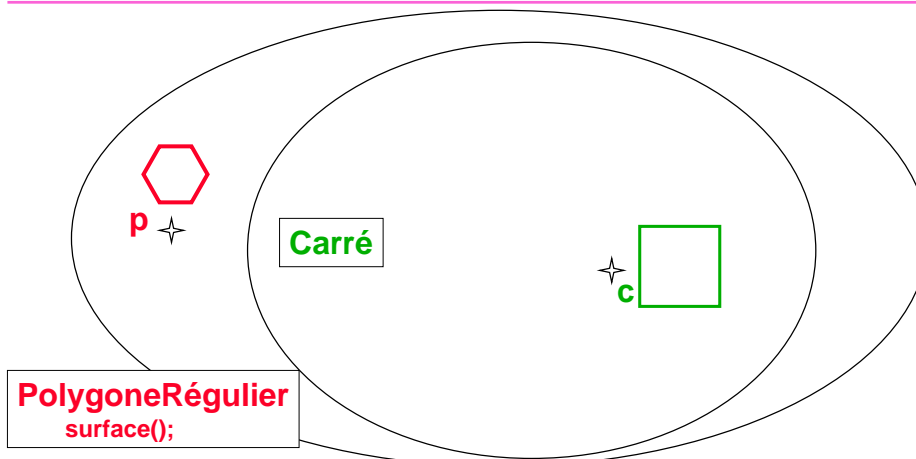
- définir une nouvelle classe en ajoutant de nouvelles fonctionnalités à une classe existante
 - ajout de nouvelles fonctions
 - ajout de nouvelles données
 - redéfinition de certaines propriétés héritées (masquage)
- Une approche de la classification en langage naturel
- Les carrés **sont** des polygones réguliers (*ce serait l'idéal...*)



ESIEE

19

Héritage

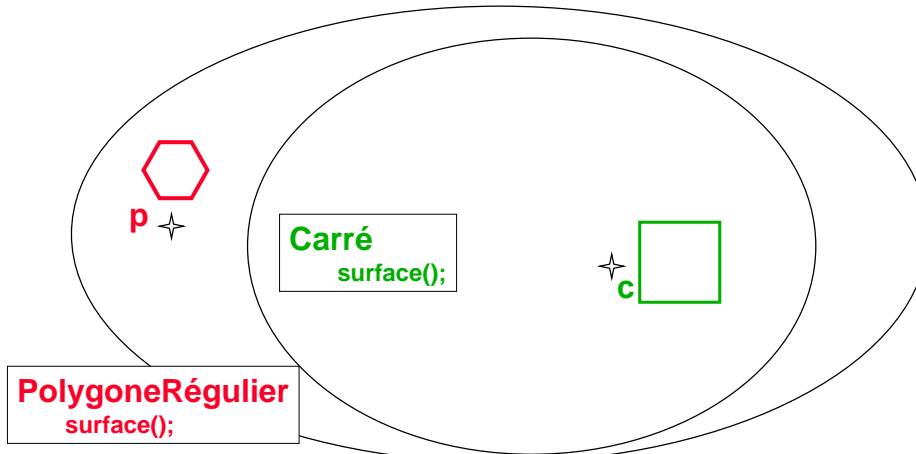


- `PolygoneRégulier p ... int s1 = p.surface();`
- `Carré c ... int s = c.surface();` ← `surface` est héritée

ESIEE

20

Héritage et redéfinition (masquage)

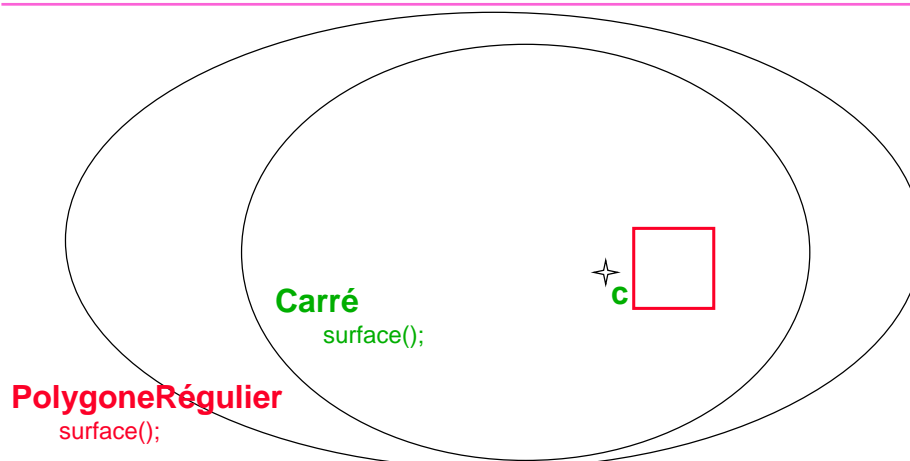


- **PolygoneRégulier p ... int s1 = p.surface();**
- **La méthode *surface* est redéfinie pour les Carrés**
- **Carré c ... int s = c.surface();**

ESIEE

21

Comportement d'un objet et méthodes



- **PolygoneRégulier p = c;**
- **p.surface();** ??? **surface();** ou **surface();** ???
-

ESIEE

22

Liaison dynamique

- **PolygoneRégulier p = c;**
- **p.surface();** ??? **surface();** ou **surface();** ???

- Hypothèses

- 1) La classe Carré hérite de la classe PolygoneRégulier
- 2) La méthode surface est redéfinie dans la classe Carré

- **PolygoneRégulier p = c;**

- **p doit se comporter comme un Carré**

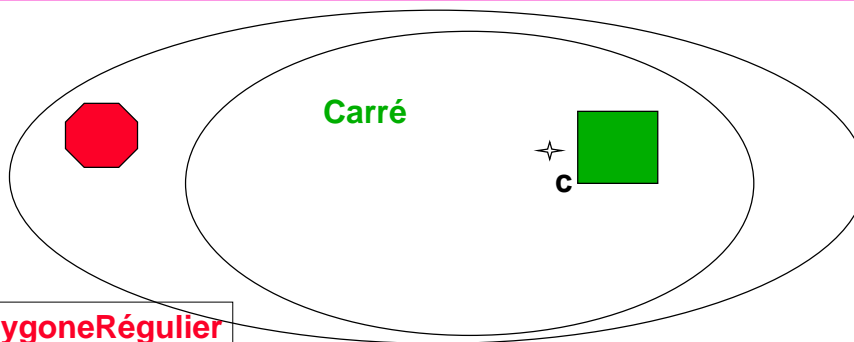
PolygoneRégulier p1 = ... un pentagone ;

- **p = p1** Possible ??? **p.surface() ???**

ESIEE

23

Les carrés sont des polygones, attention



- **Un extrait du cahier des charges**

- Les carrés sont vertes
- Les polygones réguliers sont rouges

- **? Couleur d'un PolygoneRegulier de 4 côtés ?**

- Discussion :

- Serait-ce une incohérence du cahier des charges ?
- Ou bien un polygone régulier de 4 côtés ne devrait-il pas muter en carré ?
- ...

ESIEE

24

Polymorphisme : définitions

- **Polymorphisme ad'hoc**
 - Surchage(**overloading**),
 - plusieurs implémentations d'une méthode en fonction des types de paramètres souhaités, le choix de la méthode est **résolu statiquement** dès la compilation
 - **Polymorphisme d'inclusion**
 - Redéfinition, masquage (**overriding**),
 - est fondé sur la relation d'ordre partiel entre les types, relation induite par l'héritage. si le type B est inférieur selon cette relation au type A alors on peut passer un objet de type B à une méthode qui attend un paramètre de type A, le choix de la méthode est **résolu dynamiquement** en fonction du type de l'objet receveur
 - **Polymorphisme paramétrique** ou généricité,
 - consiste à définir un modèle de procédure, ensuite incarné ou instancié avec différents types, ce choix est **résolu statiquement**
- extrait de M Baudouin-Lafon. La Programmation Orientée Objet. ed. Armand Colin

ESIEE

25

Polymorphisme ad'hoc

- $3 + 2$ $3.0 + 2.5$ "bon" + "jour"
- `out.print(5);` `out.print(5.5);` `out.print("bonjour");`
 - le choix de la méthode est **résolu statiquement** à la compilation
 - `print(int)` `print(double)` `print(String)`

ESIEE

26

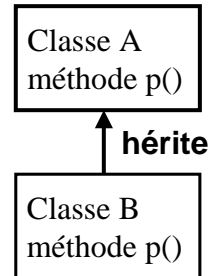
Polymorphisme d'inclusion

- `A a = new A(); a.p();`
- `B b = new B(); b.p();`
- `a = new B(); a.p();`

```
void m(A a){  
    a.p();  
}
```

```
m(new B());  
m(new A());
```

- B hérite de A, B est inférieur selon cette relation au type A
- le choix de la méthode est **résolu dynamiquement** en fonction du type de l'objet receveur



ESIEE

27

Polymorphisme paramétrique

- Une liste homogène

```
• Class Liste<T>{  
    void add(T t) ...  
    void remove(T t) ...  
    ...  
}
```

- `Liste<Integer> li = new Liste<Integer>();`
- `li.add(new Integer(4));`

- `Liste<A> la = new Liste<A>();`
- `la.add(new A());`
- `la.add(new B());`

- incarné ou instancié avec différents types, ce choix est **résolu statiquement**

ESIEE

28

Affectation polymorphe

- **Création d'instances**
 - Carre c1 = new Carre(100);
 - Carre c2 = new Carre(10);
 - PolygoneRegulier p1 = new PolygoneRegulier(4,100);
- **Affectation**
 - c1 = c2; // *synonymie* : c1 est un autre nom pour c2
 // c1 et c2 désignent un carré de longueur 10
- **Affectation polymorphe**
 - p1 = c1;
- **Affectation et changement de classe**
 - c1 = (Carre) p1; // Hum, Hum ...
 - If (p1 instanceof Carre) c1 = (Carre)p1; // mieux, beaucoup mieux ...

ESIEE

29

Liaison dynamique

- Sélection de la méthode en fonction de l'objet receveur
- **déclaré / constaté** à l'exécution
- PolygoneRegulier p1 = new PolygoneRegulier(5,100);
 // p1 **déclarée** PolygoneRegulier
- Carre c1 = new Carre(100);
- int s = p1.surface(); // p1 **constatée** PolygoneRegulier
- p1 = c1; // affectation polymorphe
- s = p1.surface(); // p1 **constatée** Carre
- Note: la recherche de la méthode s'effectue uniquement dans l'ensemble des méthodes masquées associé à la classe dérivée
 - Rappel : Dans une classe dérivée, la méthode est masquée seulement si elle possède exactement la même signature

ESIEE

30

Selon Liskov cf. Sidebar2.4, page 27

- The **apparent type** of a variable is the type understood by the compiler from information available in declarations. The **actual type** of an Object is its real type -> the type it receives when it is created.

Ces notes de cours utilisent

- *type déclaré pour **apparent type** et*
- *type constaté pour **actual type***

ESIEE

31

En pratique... <http://lmi17.cnam.fr/~barthe/OO/typage-java-2/>

```
• class A{
•   void m(A a){ System.out.println(" m de A"); }
•   void n(A a){ System.out.println(" n de A"); }
• }

• public class B extends A{

•   public static void main(String args[]){
•     A a = new B();
•     B b = new B();

•     a.m(b);
•     a.n(b);
•   }

•   void m(A a){ System.out.println(" m de B"); }
•   void n(B b){ System.out.println(" n de B"); }
• }
```

- Exécution de B.main : Quelle est la trace d'exécution ?

•m de B
•n de A

ESIEE

32

En pratique : une explication

• mécanisme de liaison dynamique en Java :

- La liaison dynamique effectue la sélection d'une méthode en fonction du type constaté de l'objet receveur, la méthode doit appartenir à l'ensemble des méthodes masquées,
- (la méthode est masquée dans l'une des sous-classes, si elle a exactement la même signature)

- Sur l'exemple,

- nous avons uniquement dans la classe B la méthode `m(A a)` masquée

- en conséquence :

- `A a = new B();` // `a` est déclarée `A`, mais constatée `B`

- `a.m` --> sélection de `((B)a).m(...)` car `m` est bien masquée

- `a.n` --> sélection de `((A)a.n(...))` car `n` n'est pas masquée dans `B`

Choix d'implémentation de Java : compromis vitesse d'exécution / sémantique ...

ESIEE

33

Un exemple en Java : la classe Carre

```
public class Carre extends PolygoneRegulier{
    private int longueurDuCote;

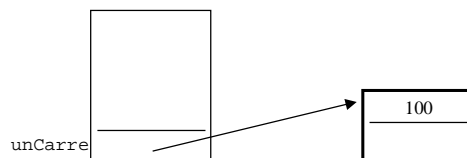
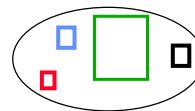
    public void initialiser(int longueur){
        longueurDuCote = longueur;
    }

    public int surface(){
        return longueurDuCote * longueurDuCote;
    }

    public int perimetre(){
        return 4*longueurDuCote;
    }
}
```

// un usage de cette classe

```
Carre unCarre = new Carre();
unCarre.initialiser(100);
int y = unCarre.surface();
```



ESIEE

34

Liskov suite [Sidebar2.4,page 27]

- **Hiérarchie**
- Java supports *type hierarchy*, in which one type can be the **supertype** of other types, which are its **subtypes**. A subtype's objects have all the methods defined by the supertype.
- **Object la racine de toute classe**
- All objects type are subtypes of **Object**, which is the top of the type hierarchy. **Object** defines a number of methods, including equals and toString. Every object is guaranteed to have these methods.

ESIEE

35

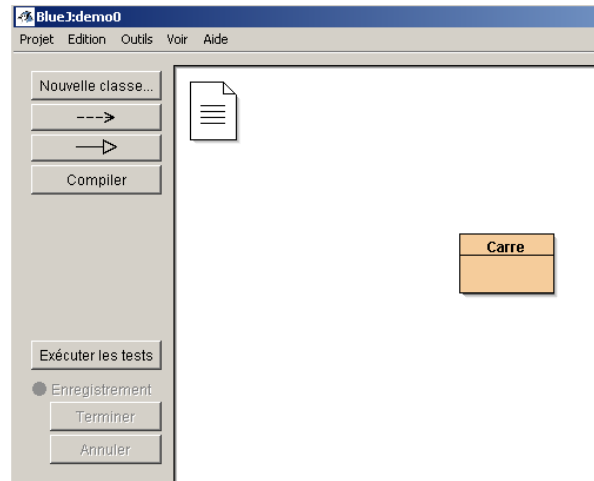
Démonstration

- **Outil Bluej**
 - La classe Carré
 - Instances inspectées
- **Tests Unitaires**
 - La Classe CarréTest
 - Ou la mise en place d'assertions pertinentes
 - En assurant ainsi des test de non régression,
 - le système ne se dégrade pas à chaque modification...
 - Augmenter le *taux de confiance* envers le code de cette classe ...
 - Très informel, et comment obtenir des tests pertinents ?
- **Tests Unitaires « référents » : Outil JNEWS**
 - Java New Evaluation Web System

ESIEE

36

Demo : Bluej

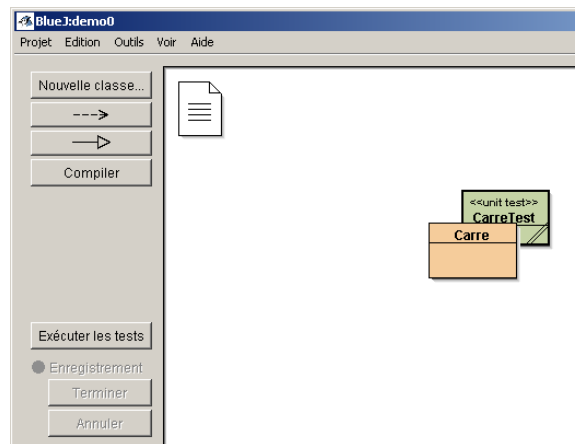


- Instances et leur inspection

ESIEE

37

Demo : Bluej + tests unitaires



- Test unitaires depuis BlueJ ou en source

ESIEE

38

Tests unitaires : outil *junit* intégré

- www.junit.org
- <http://junit.sourceforge.net/javadoc/junit/framework/Assert.html>
- Un exemple :

```
public class CarreTest extends junit.framework.TestCase{

    public void testDuPerimetre(){
        Carre c = new Carre();
        c.initialiser(10);
        assertEquals(" périmètre incorrect ???" ,40, c.perimetre());
    }
}
```

ESIEE

39

Assertions

```
assertEquals(" un commentaire ???" ,attendu, effectif);
assertSame(" un commentaire ???" ,attendu, effectif);
assertTrue(" un commentaire ???" ,expression booléenne);
assertFalse(" un commentaire ???" , expression booléenne);
assertNotNull(" un commentaire ???" , une référence);
...
```

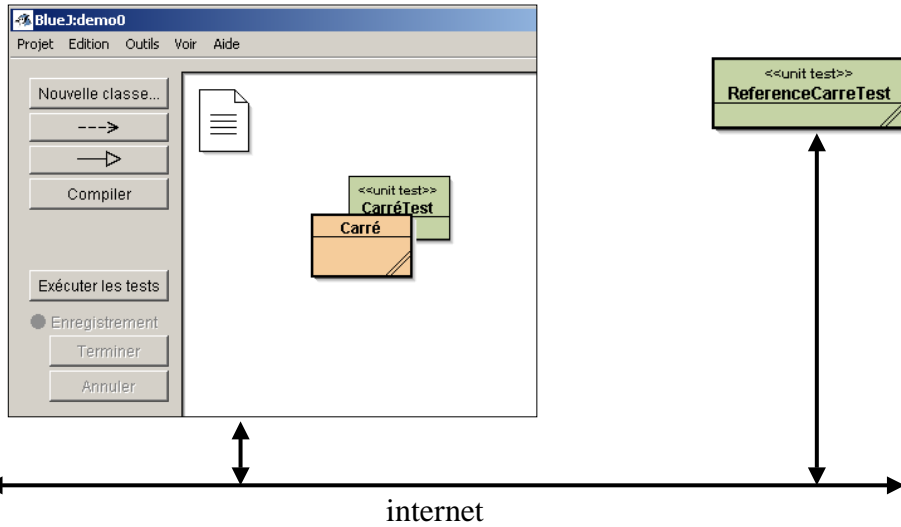
Le commentaire est affiché lorsque l'assertion échoue

ESIEE

40

JNEWS contient des Tests unitaires distants

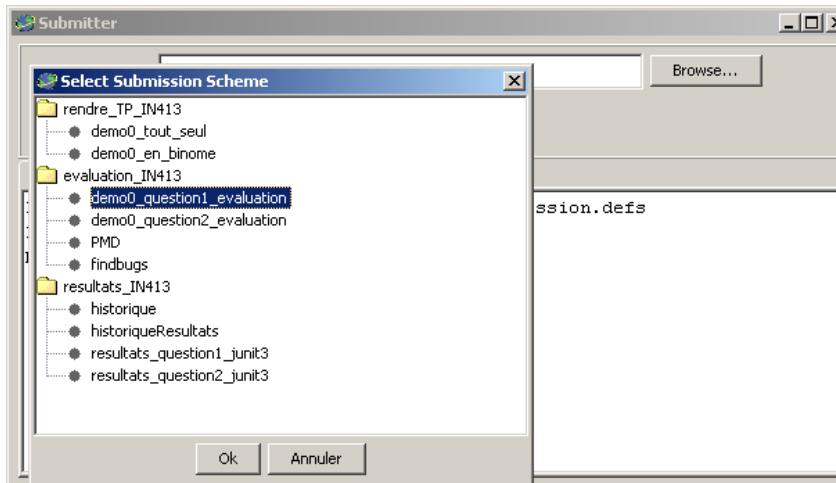
- **Tests unitaires distants et référents** (ce qui est attendu...)



ESIEE

41

JNEWS : outil *Submitter...* intégré

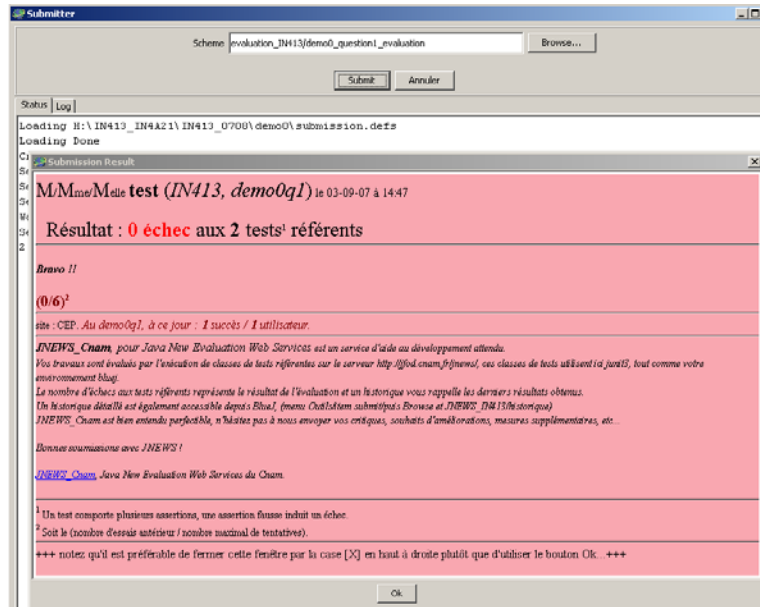


- **Un clic suffit**

ESIEE

42

Demonstration : JNEWS une réponse : Bravo!!



ESIEE

43

JNEWS à l'ESIEE, en intranet

- Une aide à la réponse attendue
 - +
 - Outils en ligne comme PMD , findbugs , checkstyle
 - +
 - Remise planifiée des sources correctement documentées
- <http://jod.cnam.fr/jnews/>

ESIEE

44

Design Pattern pourquoi faire ?

- **Patterns ou Modèles de conception réutilisables**
- **Un modèle == plusieurs classes == Un nom de Pattern**
 - > Assemblage de classes pour un discours plus clair, concis
 - > Réutilisation de savoir faire
- **Les librairies standard utilisent ces Patterns**
 - L'API AWT **utilise le patron/pattern composite ???**
 - Les évènements de Java utilisent le **patron Observateur ???**
 - ...
 - etc. ...
- **Une application = un assemblage de plusieurs patterns**