```
enum Civ {
    UKN, MR, MRS, MS
}
```

classe synthétisée par le compilateur

```
final class Civ extends Enum<Civ> {
    public static final Civ UKN;
    public static final Civ MR;
    public static final Civ MRS;
    public static final Civ MS;

    private static final Civ[] ENUM_VALUES;

    static {
        UKN = new Civ("UKN", 0);
        MR  = new Civ("MR",  1);
        MRS = new Civ("MRS", 2);
        MS  = new Civ("MS",  3);
        ENUM_VALUES = new Civ[] {
            UKN, MR, MRS, MS
        };
    }
    ...
    private Civ(String n, int r) {
        super(n, r);
    }
    public static Civ[] values() {
        Civ[] a = ENUM_VALUES;
        int n = a.length;
        Civ[] a1 = new Civ[n];
        System.arraycopy(a, 0, a1, 0, n);
        return a1;
    }
    public static Civ valueOf(String s) {
        return Enum.valueOf(Civ.class, s);
    }
}
```

```
public enum Civ {
    UKN("inconnu"),
    MR("Monsieur"),
    MRS("Madame"),
    MS("Mademoiselle");
    private final String expand;
    private Civ(String x) {
        expand = x;
    }
    @Override public String toString() {
        return expand;
    }
    public String welcome() {
        return "Bienvenue " + this;
    }
}
```

```
final class Civ extends Enum<Civ> {
    public static final Civ UKN;
    public static final Civ MR;
    public static final Civ MRS;
    public static final Civ MS;
    static {
        UKN = new Civ("UKN", 0, "inconnu");
        MR  = new Civ("MR",  1, "Monsieur");
        MRS = new Civ("MRS", 2, "Madame");
        MS  = new Civ("MS",  3, "Mademoiselle");
        ENUM_VALUES = new Civ[] {
            UKN, MR, MRS, MS
        };
    }
    ...
    private final String expand;
    private Civ(String n, int r, String x) {
        super(n, r);
        expand = x;
    }
    public String toString() {
        return expand;
    }
    public String welcome() {
        return "Bienvenue " + this;
    }
}
```

```
for (Civ c : Civ.values()) {
    System.out.println(c.welcome());
}
```

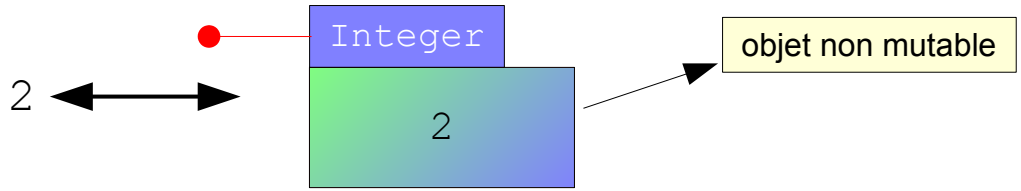
```
Bienvenue inconnu
Bienvenue Monsieur
Bienvenue Madame
Bienvenue Mademoiselle
```

*Classe enveloppante* : classe qui réifie l'un des 8 types primitifs.

*Boxing* : conversion automatique d'une valeur primitive en une instance de la classe enveloppante appropriée.

*Unboxing* : conversion automatique inverse d'un boxing.

| | |
|---|---|
| `boolean` | → `Boolean` |
| `byte` | → `Byte` |
| `short` | → `Short` |
| `int` | → `Integer` |
| `long` | → `Long` |
| `char` | → `Character` |
| `float` | → `Float` |
| `double` | → `Double` |




```
Integer x...
int n...
x = Integer.valueOf(n);
n = x.intValue();
```

à partir de Java 5

```
Integer x...
int n...
x = n;
n = x;
```

*boxing* : `int` → `Integer`

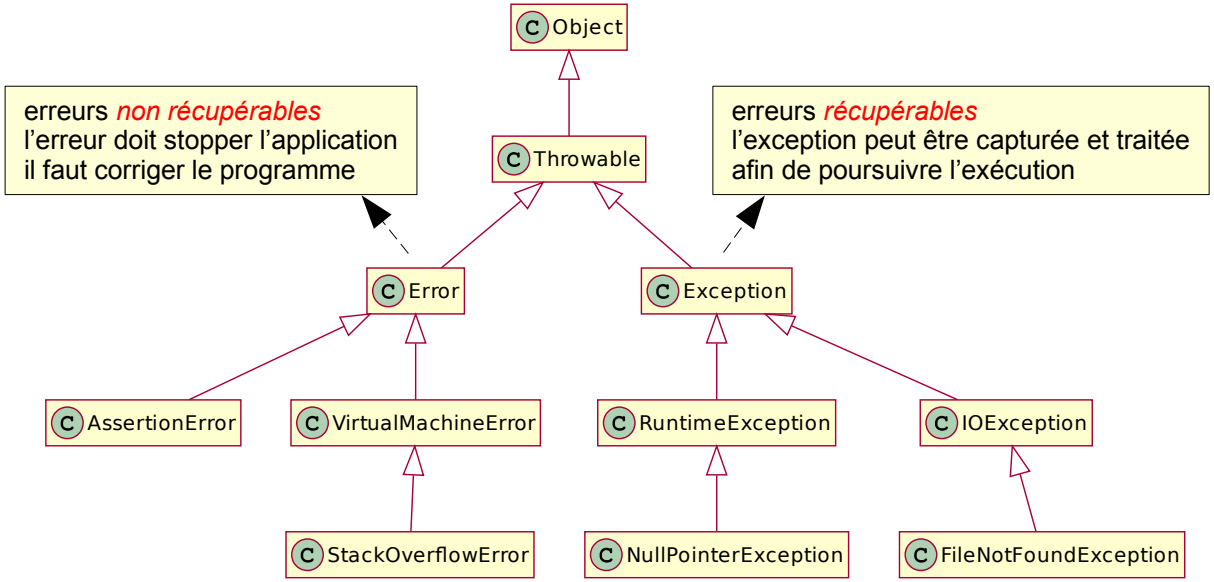
*unboxing* : `Integer` → `int`

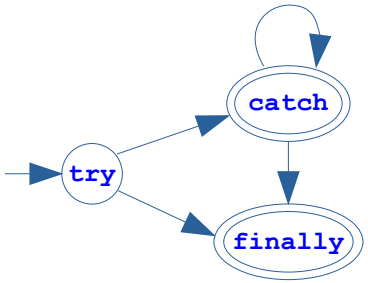
```
Integer m() {
    Integer n = 3;
    Integer m = 5;
    Integer res = 0;
    if (test) {
        res = m + n;
    } else {
        res = m - n;
    }
    return res;
}
```

meilleures performances

```
Integer m() {
    int n = 3;
    int m = 5;
    int res = 0;
    if (test) {
        res = m + n;
    } else {
        res = m - n;
    }
    return res;
}
```

*Panne logicielle* : situation anormale survenant pendant l'exécution, provoquant la fin abrupte d'une instruction.



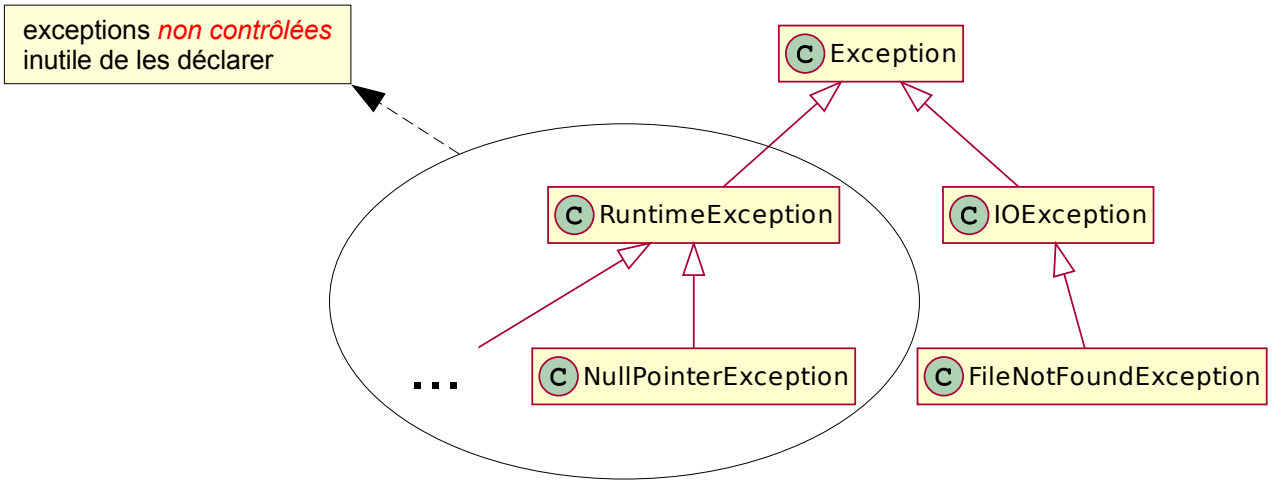




```
try {
    // code pouvant lever des exceptions
} catch (TypeException e) {
    // capture d'une TypeException
    // et gestion du problème
} catch (AutreTypeException e) {
    // capture d'une AutreTypeException
    // et gestion du problème
} finally {
    // nettoyage avant de rendre la main
    // partie exécutée qu'il y ait eu ou non
    // levée ou capture d'exception
}
```

sens de lecture des `catch`

Rappels et compléments






```java
public void truc(Object obj) {
    if (obj == null) {
        throw new NullPointerException();
    }
    ...
}
```

```java
public void write(String str) throws IOException {
    if (detectIOFailure()) {
        throw new IOException();
    }
    ...
}
```

```java
try {
    // ...
} catch (IOException e) {
    // FileNotFoundException
    // capturée ici
} catch (FileNotFoundException e) {
    // clause jamais atteinte !
}
```


```java
try {
    // ...
} catch (FileNotFoundException e) {
    // FileNotFoundException
    // capturée ici
} catch (IOException e) {
    // IOException
    // capturée ici
}
```

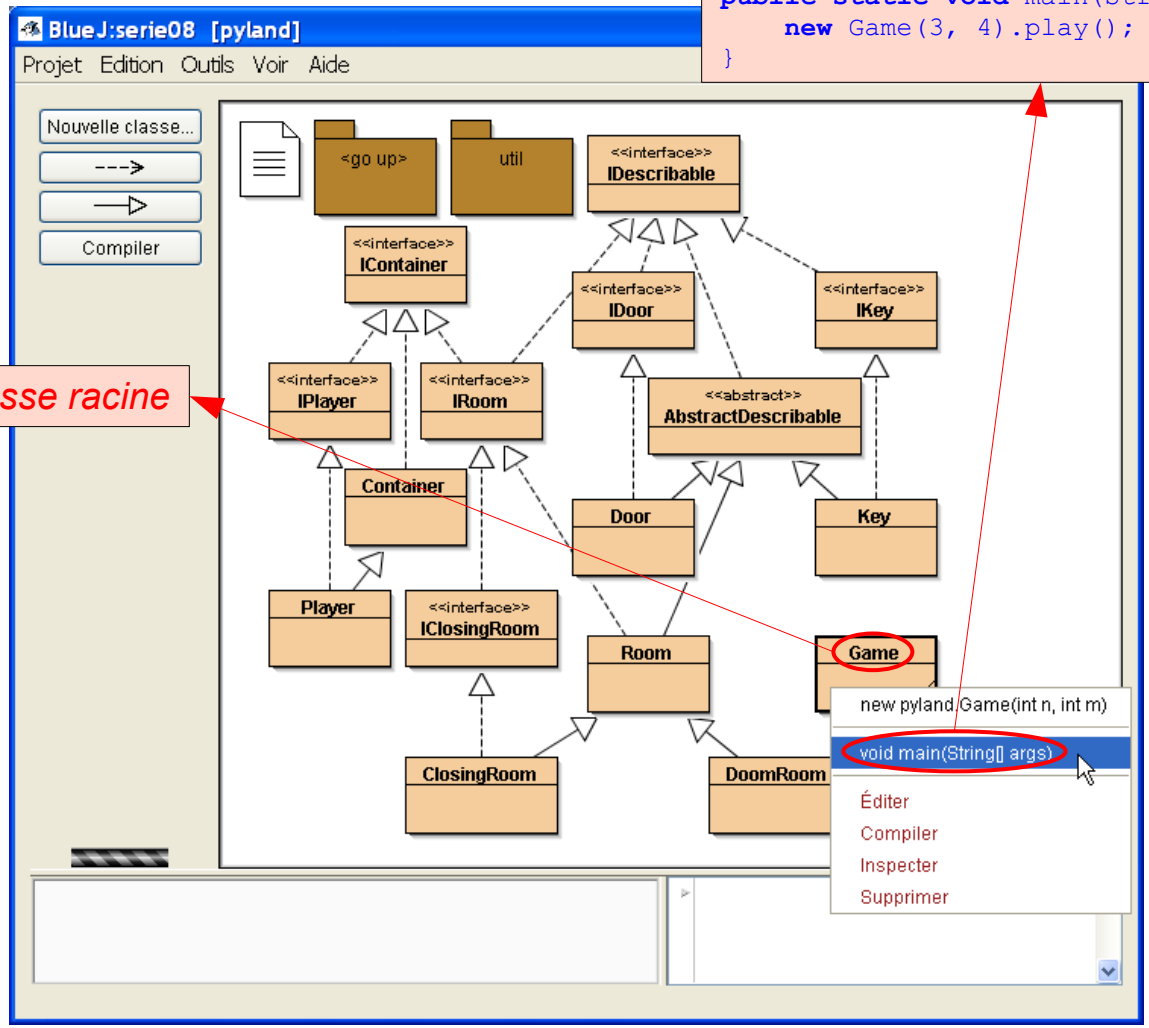
## Rappels et compléments



*Point d'entrée* :

```
public static void main(String[] args) {
    new Game(3, 4).play();
}
```

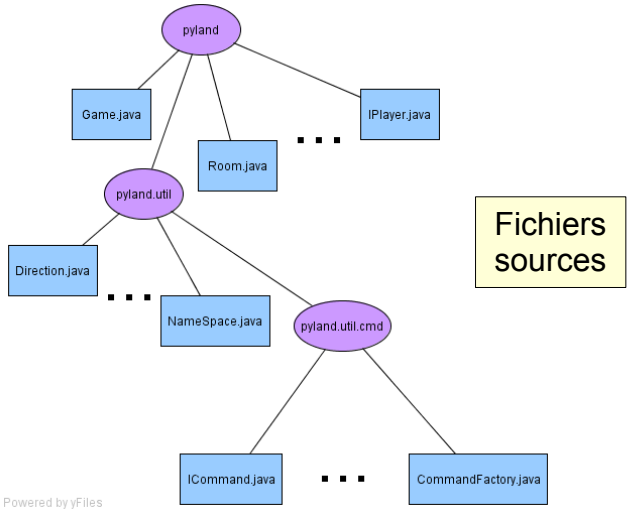
*Classe racine*




*Exemple d'application Java : PyLand (L2 info)*

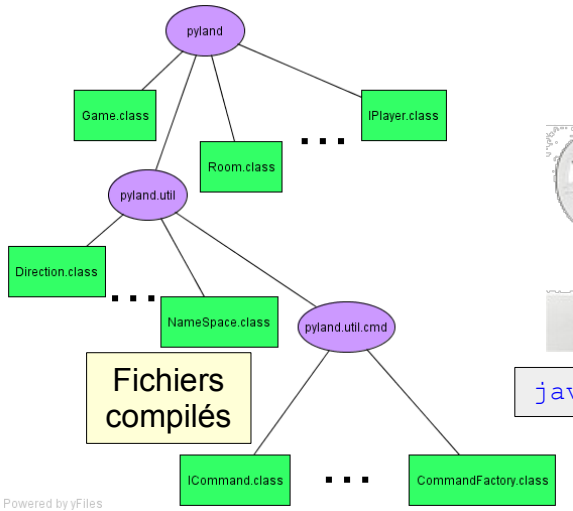
## Rappels et compléments






Fichiers sources

```
javac pyland/Game.java
```




Fichiers compilés

```
java pyland.Game
```

```
Vous êtes dans la_pièce_3
issues : north
pyland > go north
Go : Vous entrez dans la_pièce_0
issues : south east
pyland > go east
Go : Vous entrez dans la_pièce_1
issues : east west
pyland > go east
Go : Vous entrez dans la_pièce_2
issues : south west
Bravo ! Vous avez gagné !
Vous avez échappé à PY le maléfique !
```

Rappels et compléments



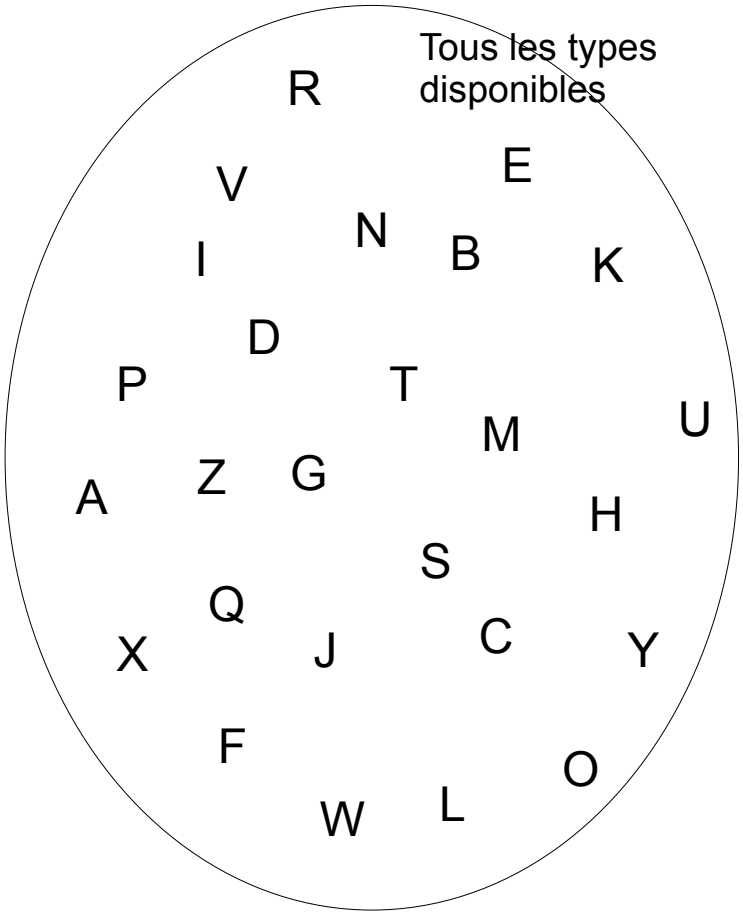
```
import A;
public class R {
    ...
    B v = new C()
    ...
}
```
→ A B C

```
import D;
class A extends E {
    ...
    B v = new F()
    ...
}
```
→ B D E F

```
import D;
class B {
    ...
    G m(D) { ... }
    ...
}
```
→ D G

```
class C extends B {
    ...
    G m(D) { ... }
    ...
}
```
→ B D G

et on continue avec D, E, F, G,...
jusqu'à la stabilité de l'ensemble
des types de l'application

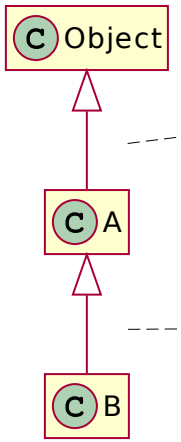



Types de l'application

Rappels et compléments



*Héritage* : mécanisme permettant la réutilisation du code de types existants, lors de la définition d'un nouveau type.




- `A` **N'HÉRITE PAS** :
  du constructeur de `Object`
- `A` **HÉRITE** de toutes les méthodes de `Object`
- `A` **DÉFINIT** de nouvelles caractéristiques

- `B` **N'HÉRITE PAS** :
  des constructeurs de `A`
  des caractéristiques statiques de `A`
  des caractéristiques de `A` inaccessibles dans `B`
- `B` **HÉRITE** de toutes les autres caractéristiques de `A`
- `B` **DÉFINIT** de nouvelles caractéristiques

```
public class Object {
    public Object() { /* rien */ }
    public boolean equals(Object) { ... }
    public final Class getClass() { ... }
    public int hashCode() { ... }
    public String toString() { ... }
    protected Object clone() throws CNSE { ... }
    protected void finalize() throws Throwable { ... }
    public final void notify() { ... }
    public final void notifyAll() { ... }
    public final void wait() throws IE { ... }
    public final void wait(long to) throws IE { ... }
    public final void wait(long to, int n) throws IE { ... }
}
```

```
class A {
    public static final int C = 0;
    private int i;
    public void m() { p(); }
    private void p() { ... }
    public static void ms() { ... }
}
```

A hérite de
- toutes les méthodes de `Object`

A définit
- les attributs `i` et `C`
- le constructeur `A()`
- les méthodes `m()`, `p()` et `ms()`

```
class B extends A {
    private int j;
    public void q() { ... }
}
```

B hérite de
- toutes les méthodes de `Object`
- la méthode `m()` de `A`

B définit
- l'attribut `j`
- le constructeur `B()`
- la méthode `q()`