L'environnement *oRis*

(version 2.3 — 25 juin 2001)

Les principes
Le langage
Les services

Fabrice Harrouet
Laboratoire d'Informatique Industrielle
Ecole Nationale d'Ingénieurs de Brest
`harrouet@enib.fr`
`http://www.enib.fr/~harrouet/`

Les principes

- ▷ **Propos de l'outil**
- ▷ **Un langage de programmation**
- ▷ **Un environnement de simulation**
- ▷ **Services utilitaires**

Propos de l'outil

- ▷ **Modéliser et ajuster un système interactivement**
  - ◇ Système modifiable dans le temps
  - ◇ Interaction par le langage, pendant le fonctionnement
- ▷ **Composants du système $\equiv$ entités autonomes**
  - ◇ Pas de modèle global $\rightarrow$ modularité
  - ◇ Minimiser l'influence de l'ordonnanceur
- ▷ **Traiter des thèmes variés**
  - ◇ Langage généraliste
  - ◇ Services utilitaires facilement utilisables

$\Rightarrow$ **Outil de prototypage interactif**

Un langage de programmation

- **Interprété, entièrement dynamique**
  - Introduire du code à tout instant
  - Influer sur le déroulement de l'application (au delà d'une IHM)
  - Ajouter de nouvelles notions
  - Modifier l'existant
- **Orienté objet**
  - Formalisme *"habituel"*
  - Adapté aux entités autonomes
- **Fortement typé**
  - Détection des erreurs au plus tôt
  - Erreurs potentielles plutôt que erreurs effectives

Un environnement de simulation

- **Entités autonomes**
  - Notion d'objets actifs
  - Moyens de communication variés
- **Multi-tâches**
  - Évolution des comportements en parallèle
  - Ordonnanceur équitable, propriétés
  - Résultat $\equiv$ conséquence des choix algorithmiques
- **Environnement géométrique**
  - Objets situés et visualisateurs en 2D/3D
  - Moyens de perception

Services utilitaires

- **Bibliothèques variées**
  - Aborder de nombreux sujets applicatifs
  - Concernant la simulation
  - Concernant des points techniques
- **Facilité d'emploi**
  - Peu de paramètres
  - Utilisation immédiate par l'utilisateur

Le langage

- ▷ **Généralités**
- ▷ **Les fonctions**
- ▷ **Les objets**
- ▷ **Le parallélisme**
- ▷ **La communication**
- ▷ **La dynamicité**
- ▷ **Le ramasse-miettes**
- ▷ **Les moyens d'extension**

Généralités

▷ **Langage interprété, entièrement dynamique**

- Impératif, procédural, orienté-objets, multi-tâches

▷ **Syntaxe et concepts proches de *C++* et *Java***

- Déclarations de variables
- Instructions / expressions
- Structures de contrôle usuelles ...

▷ **Prise en main facile *"pour un programmeur"***

Généralités

- **Langage fortement typé : types de base**
  - `int` : entier signé 32 bits
  - `float` : réel double précision
  - `string` : chaîne de caractères (dynamiquement extensible)
  - *NomDeClasse* : référence sur un objet (instanciation dynamique)
  - ***Type*** `[]` : tableau (multi-dimensions, dynamiquement extensible)
- **Tous ont une valeur logique**
  - Entier nul `0` $\rightarrow$ *faux*
  - Réel nul `0.0` $\rightarrow$ *faux*
  - Chaîne vide `""` $\rightarrow$ *faux*
  - Référence nulle `(`***Type***`)NONE` $\rightarrow$ *faux*
  - Tableau vide `(`***Type***`[])[]` $\rightarrow$ *faux*

Généralités

- **Opérateurs sur les réels**
  - `=   ++   +=   +   --   -=   -   *=   *   /=   /   %=   %`
  - `==   !=   <?   >?   <   >   <=   >=`
- **Opérateurs sur les entiers**
  - Ceux des réels
  - `~   &=   &   |=   |   ^=   ^   <<=   <<   >>=   >>`
- **Opérateurs sur les chaînes**
  - `=   +=   +   ==   !=   <?   >?   <   >   <=   >=`
  - `s=s[i];  i=s1.seek(s2);  nb=s.length();`
  - `str=valueDump(anyData);  valueParse(anyData,str);`
  - `"Les chaînes tiennent sur plusieurs lignes sans utiliser le caractère de neutralisation \\"`

Généralités

- **Opérateurs sur les tableaux**
  - `=  +=  +  ==  !=  <?  >?  <  >  <=  >=`
  - ```
    t.insert(d); t.pushFront(d); t.pushBack(d);
    d=t.remove(i); d=t.popFront(); d=t.popBack();
    t.erase(); nb=t.length(); d=t[i]; t[i]=d;
    d=t.front(); t.front()=d; d=t.back(); t.back()=d;
    t.sort(); i=t.seekMin(); i=t.seekMax(); i=t.seek(d);
    ```
- **Opérateurs sur les reférences/instances**
  - `=  ==  !=  new  delete`
  - ```
    :=  :==  :!=  <?  >?  <  >  <=  >=  i=t.deepSeek(d);
    bool Object::affect(Object obj);
    bool Object::equalsTo(Object obj);
    bool Object::lessThan(Object obj);
    ```

Généralités

- **Autres opérateurs**
  - Opérateurs logiques :
    - `!   &&   ||   ?:`
  - Tirage aléatoire :
    - Réel dans $[0.0; 1.0]$: `r=?;`
    - Entier dans $[0; nb-1]$: `i=?(nb);`
- **Remarque**
  - Les chaînes et tableaux sont des types de base
  - Utilisation d'un `.` pour les opérateurs

Généralités

▷ **Les directives**

◇ Inclusion de fichiers

```
include "stuff.pkg"
```

◇ *"Sentinelles"*

```
beginPackage STUFF_PKG ...  endPackage
```

◇ Substitutions lexicales

```
replace HALF_PI by 1.5707963;
replace PRIVATE by if(that!=this) error("Private");
```

▷ **Les commentaires**

◇ `// Le reste de la ligne est un commentaire`

◇ `/* Les commentaires /* peuvent etre */ imbriques */`

Généralités

▷ **Exécuter du code**

◇ Fonction anonyme interruptible

```
start{/* code a executer */}
```

◇ Fonction anonyme non interruptible

```
execute{/* code a executer */}
```

◇ Méthode anonyme interruptible

```
Stuff.4::start{/* code a executer par Stuff.4 */}
```

◇ Méthode anonyme non interruptible

```
Stuff.4::execute{/* code a executer par Stuff.4 */}
```

◇ (Voir les objets et le parallélisme pour les nuances)

◇ Évoque la fonction `main()` de *C/C++*

Généralités

▷ **Les entrées/sorties**

◇ Écriture dans la console

```
print(e1,e2,...);  println(e1,e2,...);
```

◇ Lecture depuis la console

```
str=readln();
```

◇ Écriture dans une chaîne

```
str=format(e1,e2,...);
```

◇ Avertir d'une incohérence potentielle (affichage de la pile)

```
warning(e1,e2,...);
```

◇ Interrompre le traitement suite à une incohérence (affichage de la pile)

```
error(e1,e2,...);
```

Généralités

▷ **Utilisation de tableaux**

```
execute
{
float[] data=[1.2,3.4,5.6,7.8]; // un tableau initialise
float[][] m; // une matrice, vide pour l'instant
for(int raw=0;raw<data.length();raw++)
  {
  m.pushBack((float[])[]); // ajouter une ligne vide
  for(int column=0;column<data.length();column++)
    {
    m.back().pushBack(raw*data[column]); // remplir la ligne
    }
  }
}
```

Les fonctions

- **Déclaration** $\equiv$ *C/C++*

```
void functionA();
int functionB(void);
TypeRetour nomFonction(Type arg1, Type arg2,...);
```

- **Déclaration obligatoire si définition récursive**
  - Analyse correcte $\rightarrow$ fonction prise en compte
  - ```
    int fact(int n); // Declaration prealable
    int fact(int n) // Definition
    {
    return(n<=1 ? 1 : n*fact(n-1)); // appel recursif
    }
    ```
    ```

Les fonctions

▷ **La surcharge est interdite**

◇ Un nom de fonction → une seule signature

▷ **Passages par référence**

◇ Uniquement pour les paramètres

◇ Utilisation de `&`

```
void cartToPol(float & d,float & a,float x,float y);
```

▷ **Variables locales à <u>toute</u> la fonction**

◇ Initialisées à `0`, `0.0`, `""`, `(`*`Type`*`)NONE`, `(`*`Type`*`[])[]`

Les objets

- **Classe de base :** `Object`
  - Toutes les classes en hérite implicitement
- **Déclaration semblable à** *C++*

```
class MyClass : ParentA, ParentB // heritage
{
int _value; // des attributs
float[] _array;
void new(int initValue); // le constructeur
void delete(void); // le destructeur
int getValue(void); // des methodes
float[] getArray();
}
```

Les objets

- **Définition des méthodes**

```
float[] MyClass::getArray(void)
{
return(_array);
}
```

- **Propriétés des méthodes ≡ propriétés des fonctions**
  - Pas de surcharge
  - Paramètres par référence avec `&`
  - Variables locales à toute la méthode, et initialisées

Les objets

- **Attributs initialisés**
  - `0`, `0.0`, `""`, `(`*Type*`)NONE`, `(`*Type*`[])[]`
- **Tout est** *public*
- **Pas de classes imbriquées**
- **Contructeur et destructeur obligatoires**
- **Instanciation exclusivement dynamique**
  - `Object o1=new Object;`
  - `Object o2=new Object();`
  - `MyClass c=new MyClass(2);`
- **Destruction** (voir le ramasse-miettes)
  - `delete c;`

Les objets

▷ **Accès aux attributs et aux méthodes**

```
println(c->_array);  c->_value=5;  c->method();
```

▷ **Appels qualifiés**

◇ Non qualifié → recherche de la méthode, puis de la fonction

◇ Appel de fonction

```
::function();
```

◇ Appel de méthode

```
obj->method();  obj->Parent::method();
```

◇ Réutilisation d'une méthode surdéfinie

```
void MyClass::method(void)
{...
Parent::method();
}
```

Les objets

▷ **Descente des constructeurs**

```
void MyClass::new(int initValue)
{
ParentA::new(initValue); // implicite si pas
ParentB::new(initValue,4); //   de parametre
}
```

▷ **Remontée des destructeurs**

```
void MyClass::delete(void)
{
ParentA::delete(); // inutile car
ParentB::delete(); //   implicite
}
```

Les objets

▷ **Définition compacte d'une classe**

```
class MyClass : ParentA, ParentB
{
int _value;
float[] _array;
void new(int initValue) { _value=initValue; }
void delete(void) { }
int getValue(void) { return(_value); }
float[] getArray(void) { return(_array); }
}
```

Les objets

▷ **Définition fragmentée d'une classe**

```
class MyClass; // declaration simplifiee
class MyClass : ParentA, ParentB ; // ajout heritage

class MyClass { int _value; } // ajout d'un attribut
float[] MyClass::_array; // ajout d'un autre attribut

void MyClass::new(int initValue) // ajout des methodes
{
ParentA::new(initValue);
ParentB::new(initValue,4);
_value=initValue;
}
...
```

Les objets

▷ **Séparation interface/implémentation**

```
//---- myClass.pkg ---- Interface de MyClass ----
beginPackage MYCLASS_PKG
class MyClass : ParentA
{
void new(int initValue);
void delete(void);
int getValue(void);
float[] getArray(void);
}
include "myClass.imp" // sauf si references croisees
endPackage // MYCLASS_PKG
```

Les objets

▷ **Séparation interface/implémentation**

```
//---- myClass.imp ---- Implementation de MyClass ----
beginPackage MYCLASS_IMP
class MyClass : ParentB // heritage cache
{
int _value; float[] _array; // attributs
void _hiddenMethod(void); // methode cachee
}
void MyClass::new(int initValue) { ... }
void MyClass::delete(void) { ... }
void MyClass::_hiddenMethod(void) { ... }
int MyClass::getValue(void) { ... }
float[] MyClass::getArray(void) { ... }
endPackage // MYCLASS_IMP
```

Les objets

▷ **Faciliter l'interaction avec les objets**

◇ Nommage automatique des instances

```
string str=obj->name();
Car c=(Car)getObject("Car.12");
```

◇ Constantes lexicales de type référence : `Avion.5` `Predator.2`

▷ **Introspection simple**

◇ Le nom des classes existantes : `string[] l=getClasses();`

◇ Le nombre d'instances d'une classe : `int nb=numberOf("Car");`

◇ Les instances d'une classe : `Car[] l=(Car[])getAll("Car");`

◇ Tester la validité d'une référence : `if(exist(obj)) ...`

◇ La classe d'une instance : `string str=obj->className();`

◇ Relations d'héritage : `if(obj->isA("Car")) ...`

◇ (voir `reflect.pkg` et `browser.pkg`)

Les objets

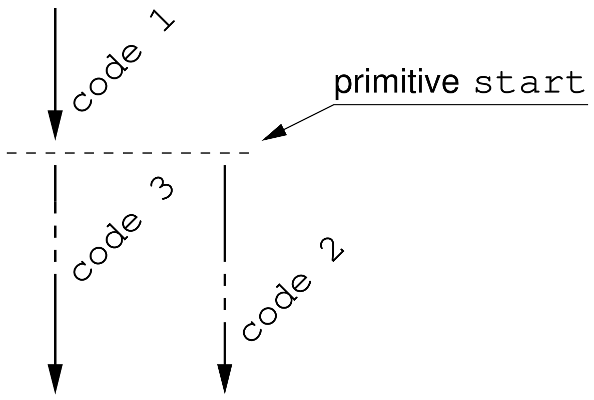
- **Objets actifs**
  - Disposent d'une méthode `void main(void)`
    - Activité principale du comportement
  - Autonomie d'exécution
    - `main()` est démarrée à l'instanciation
    - `main()` est relancée dès qu'elle est terminée
- **Application décentralisée**
  - Portion de code pour instancier des objets actifs
  - Les laisser évoluer
  - État global ≡ interaction des comportements individuels

Le parallélisme

- **Exécution de tous les flots d'instructions en parallèle**
- **Les unités d'exécution**
  - Méthode `main()` des objets actifs
  - Blocs `start` et `execute` externes
  - Primitive d'embranchement `start`

Code d'une fonction ou d'une méthode

```
{
/* code 1 */
start
  {
  /* code 2 */
  }
/* code 3 */
}
```



Le parallélisme

▷ **Contrôle des unités d'exécution**

◇ Obtenir l'identifiant d'une unité d'exécution

- `int execID=currentExecution();`
- `int execID; start(execID) { ... }`

◇ Suspendre/relancer toutes les activités d'une instance

- ```
  if(obj->isSuspended()) obj->resume();
  else obj->suspend();
  ```

◇ Suspendre/relancer une unité d'exécution

- ```
  if(isSuspended(execID)) resume(execID);
  else suspend(execID);
  ```

◇ Détruire une unité d'exécution

- `abort(execID); // relance si main()`

Le parallélisme

▷ **Cycle de simulation**

- ◇ Notion complètement artificielle ≡ temps commun
  `int clock=getClock();`
- ◇ Cycle ≡ exécution d'une **portion** de chaque unité d'exécution
- ◇ Une désignation et une seule par cycle
- ◇ Progression potentiellement équivalente pour chaque flot d'exécution
- ◇ Interruptible → parallélisme

Le parallélisme

- **Machine virtuelle oRis**
  - Ensemble de flots d'exécution
    - Pile de contextes
      - Module exécutable (fonction, méthode ...)
        - Séquence de micro–instructions
      - Variables locales
      - Compteur d'instructions
    - Pile de valeurs temporaires

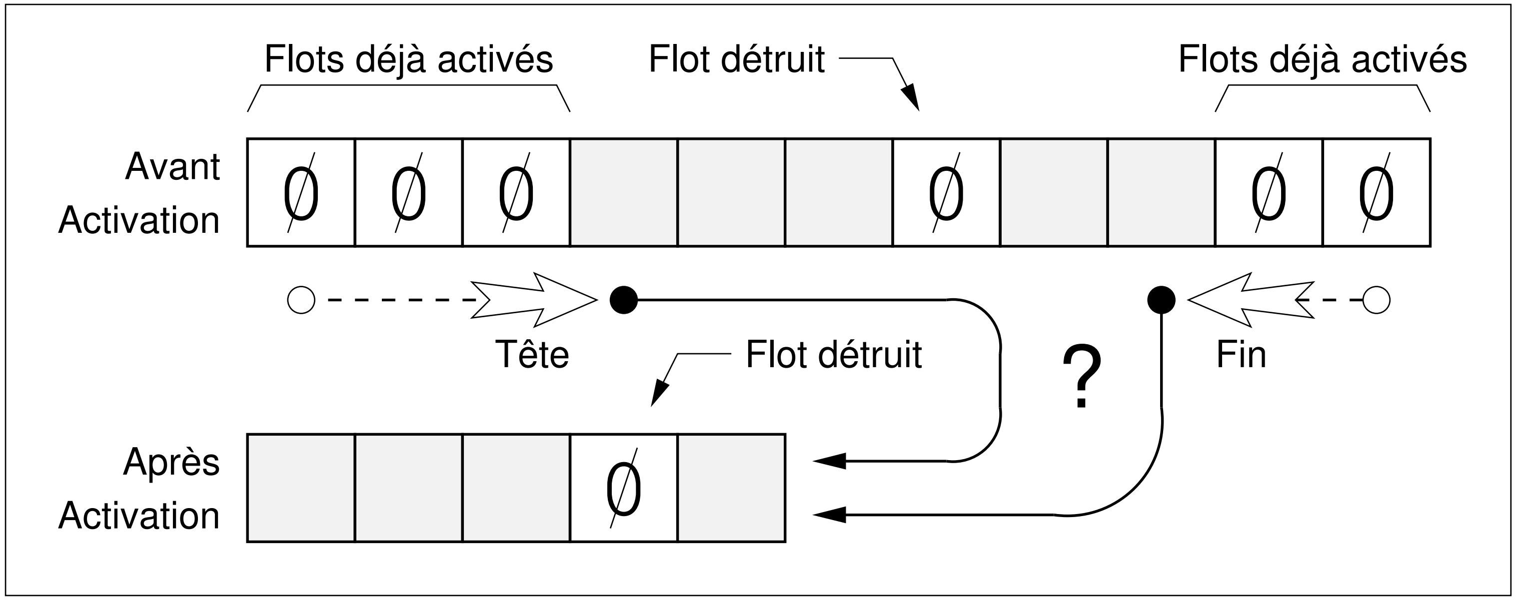
Le parallélisme

- **Multi-tâche coopératif**
  - Commutation explicite
    (`yield()`, fin de `main()`, attente de ressource ...)
  - Disponible par : `setInterval(0); // default`
- **Multi-tâche préemptif**
  - Commutation sur signal d'horloge
  - Temps du processus (`ITIMER_PROF`)
  - Disponible par : `setInterval(dt); // dt in ms`
- **Multi-tâche profondément parallèle**
  - Commutation après `N` micro-instructions
  - Debug → mise en évidence des sections critiques
  - Disponible par : `setInterval(-N); // arg < 0`

Le parallélisme

▷ **Désignation au sein d'un cycle de simulation**

◇ Deux vecteurs d'unités d'exécution à permuter



◇ Aléatoire : `setPickingMode(randomPicking); // default`

◇ Toujours en tête : `setPickingMode(serialPicking);`

Le parallélisme

- **Sections critiques de bas niveau**
  - Interdire les commutations durant une portion de code
  - Blocs `execute` externes
  - Blocs `execute` internes

```
{
/* code interruptible */
execute
  {
  /* code non interruptible */
  }
/* code interruptible */
}
```

  - Constructeurs, destructeurs, méthodes *"réflexes"* (événement d'IHM, réception d'un message . . .)

Le parallélisme

- **Exclusion mutuelle**
  - Classe `Semaphore` (`include "semaphor.pkg"`)
  - Initialisation variable de la ressource
    ```
    Semaphore s=new Semaphore(1);
    ```
  - Vérouillage/dévérouillage
    ```
    s->lock(); ... ; s->unlock();
    if(s->tryLock()) { ... ; s->unlock();}
    ```
  - Blocage des flots d'exécution attendant la ressource
  - Ordre variable de relance des flots
    ```
    s->setRandom(false); // relance FIFO (default)
    s->setRandom(true); // relance aléatoire
    ```

Le parallélisme

▷ **Erreurs à l'exécution**

◇ Non détectables à l'analyse lexicale, syntaxique ou sémantique

◇ Génération d'un message dans la console

- Nature de l'erreur
  (division par zéro, accès à un objets détruit ...)
- Pile d'exécution
  (méthodes/fonctions invoquées et objets exécutant)

◇ Destruction du flot d'exécution incriminé uniquement

- Le reste de l'application continue normalement
- Interventions possibles *"en ligne"* pour réparer

La communication entre objets

- **Communication synchrone**
  - Usuel en programmation orientée-objets
  - Invocation de méthodes : `obj->method();`
  - Traitement effectué par le flot d'exécution appelant
  - Méthode exécutée sur `this` et invoquée par `that`
  - Tout est public mais :
    - Contrôles dynamique de `that`

      ```
      if(that!=this)  if(!that->isA("AllowedClass"))
      ```
    - Contrôles dynamique de la pile d'exécution

      ```
      Object[] callers;
      string[] modules;
      int execID=currentExecution();
      getExecutionStack(callers,modules,execID);
      ```

La communication entre objets

▷ **Liens réflexes**

◇ Classe `AttrCB` (`include "attrCB.pkg"`)

◇ Connecté à un attribut d'une instance

```
AttrCB a=new AttrCB(obj,"attributeName");
```

◇ Modification de l'attribut → méthodes réflexes

```
void AttrCB::before(void);
void AttrCB::after(void);
```

◇ Détection automatique des boucles

- ∘ Attribut modifié une seule fois par réflexe
- ∘ Déconnection temporaire

```
force{ /* AttrCB sans effet */ }
```

◇ Contrôle, bornage, cohésion ...

La communication entre acteurs

- **Envoi de messages en point à point asynchrone**
  - Support : classe `Message`
    - Information minimale : émetteur (`that` dans constructeur)
  - Émission

```
Message msg=new Message;
msg->sendTo(destinataire);
```

  - Réception
    - Réflexe

```
void Object::onMessage(void) {resume();}
```

    - Consultation

```
int nb=getNbMessages();
Message msg=getNextMessage(); // NONE si aucun
```

    - Traitement ultérieur par une activité du destinataire

La communication entre acteurs

- **Diffusion de messages**
  - Support : même classe `Message` qu'en point à point
  - Émission
    `Message msg=new Message;  msg->broadcast();`
  - Abonnement
    - `setSensitivity("EventType","callbackMethod");`
    - `cancelSensitivity("EventType");`
    - Abonnements multiples et dynamiques
  - Réception
    - Réflexe adapté au type de message (selon abonnement)
    - `void Object::callbackMethod(EventType msg);`
    - Traitement ultérieur comme en point à point
      `void Object::asynchronousReception(Message msg);`

La dynamicité

- **Introduction de code à tout instant**
  - Une chaîne de caractères générée par le code existant
    `parse("..."); eval("...");`
  - Un fichier
    `load("file.ors");`
  - Une saisie dans une fenêtre de dialogue
  - Une trame réseau

- **L'intégrité du langage est disponible en ligne**
  - Forme du code strictement identique au script initial
  - Compléter l'application en cours de fonctionnement
  - L'application peut se modifier elle-même

La dynamicité

- **Interaction directe avec l'application en cours**
  - Instancier/détruire des objets
  - Invoquer des méthodes, modifier des paramètres
- **Réécriture de portions d'applications**
  - Les fonctions
  - Les méthodes au niveau des classes
  - Les méthodes au niveau des instances
    ```
    void Car.12::brake(void) { ... }
    ```
  - Remplacement si pas d'erreur à l'analyse
    - Concerne les anciennes et les nouvelles instances
    - Immédiatement utilisable
    - Les précédentes versions terminent leur exécution

La dynamicité

- **Ajout de nouvelles notions**
  - De nouvelles fonctions
  - De nouvelles méthodes au niveau des classes
  - De nouveaux attributs au niveau des classes
    `int MyClass::_newAttrib;`
  - De nouvelles méthodes au niveau des instances
    `void Car.12::park(void) { ... }`
  - De nouveaux attributs au niveau des instances
    `int Car.12::_somethingNew;`
  - Accepté si pas d'erreur à l'analyse
    - Concerne les anciennes et nouvelles instances
    - Immédiatement utilisable

La dynamicité

▷ **Quelques précautions**

◇ Constructeur/destructeur implémentés → pas de nouveaux héritages

◇ Que des réécritures et des ajouts, pas de retraits

∘ Rendre une fonction/méthode virtuelle pure

```
void Car.12::park(void); // annule la définition
```

◇ Pas de classe abstraite

∘ Ne pas invoquer les méthodes non implémentées (erreur à l'exécution)

∘ L'implémentation peut arriver plus tard

Le ramasse-miettes

- **Simple compteur de références**
  - Destruction si aucune référence sur une instance
  - L'exécution d'une méthode référence `this`
    - Les objets actifs sont auto-référencés
- **Destruction explicite toujours possible**
  - `delete anObject;`
  - Utilisation d'une référence invalide → erreur
  - Tester la validité d'une référence
    `if(exist(anObject)) ...`

Le ramasse-miettes

- **Choisir les objets concernés**
  - À l'instanciation
    - `Object obj1=new Object; // concerné`
    - `Object obj2=NEW Object; // non concerné`
  - En cours d'utilisation
    - ```
      Object obj1=new Object; // concerné
      obj1->setTransient(false); // non concerné
      ```
    - ```
      Object obj2=NEW Object; // non concerné
      obj2->setTransient(true); // concerné
      ```
  - À la conception
    - ```
      void MyClass::new(void)
      { setTransient(false); /* ... */ }
      ```
      ```

Les moyens d'extension

- **Ajout de code compilé en** *C++*
  - Chargement sous forme de *plug-in*
    - `loadLibrary("myLibrary");`
  - Définir des classes compilées (associées à des classes ***oRis***)
    - `native class MyClass { ... }`
    - Correspondance entre instances ***oRis*** et instances *C++*
    - Pas d'héritage multiple entre classes natives
  - Définir des fonctions/méthodes compilées
    - `native void f(void);`
    - Pas encore d'*API* facilement utilisable

Les moyens d'extension

▷ **Passage dynamique interprété ↔ compilé**

◇ La déclaration (prototype) est la même dans les deux cas

```
void f(void); // declaration
```

◇ Le mécanisme de surdéfinition ne modifie que la définition (distinction *interprété*/*compilé* à ce niveau)

```
void f(void) { ... } // définition
native void f(void); // définition
```

◇ Plusieurs implementations possibles

```
native "::f1()" void f(void); // une implémentation
native "::f2()" void f(void); // une autre
```

Les moyens d'extension

▷ **Invocation de programmes externes**

- ◇ `int system(string cmd);`
- ◇ `void systemI(string cmd,string input);`
- ◇ `string systemO(string cmd);`
- ◇ `string systemE(string cmd);`
- ◇ `string systemIO(string cmd,string input);`
- ◇ `string systemIE(string cmd,string input);`
- ◇ `string systemOE(string cmd,string & err);`
- ◇ `string systemIOE(string cmd,string input,string & err);`

Les moyens d'extension

- **Interaction avec le système**
  - `void systemYield(void); // process`
  - `int getTime(void); // time in ms`
  - `string getEnv(string varName);`
  - `void putEnv(string varName,string value);`
  - `string systemName(bool detailed);`
  - `string[] commandLine(bool full);`
  - `string locateFile(string fileName); // ORISPATH`
  - `void showURL(string url,bool newWindow); // Netscape`

Les services

- ▷ **Les notions prédéfinies**
- ▷ **Quelques services utilitaires**
- ▷ **Les conteneurs génériques**
- ▷ **Les composants graphiques**
- ▷ **Les moyens de réflexion**
- ▷ **Ouverture à d'autres systèmes**
- ▷ **Ouverture à d'autres langages**

Les notions prédéfinies

```
replace bool by int; //---------------------- Type logique # Type entier
replace true by 1;
replace false by 0;

class Message //---------- Support pour la communication des ‘‘acteurs’’
{
void new(void);
void delete(void);
void broadcast(void);
void sendTo(Object receiver);
Object getEmitter(void)
void setEmitter(Object emitter);
};
```

Les notions prédéfinies

```
class Object //----------------- Ancetre implicite de toutes les classes
{
void new(void);
void delete(void);
void deleteOnExit(void);
string name(void);
bool isA(string className);
string className(void);
int suffix(void);
void suspend(void);
void resume(void);
bool isSuspended(void);
void setTransient(bool isTransient);
bool isTransient(void);
...
```

Les notions prédéfinies

```
...
bool equalsTo(Object obj);
bool lessThan(Object obj);
void affect(Object obj);
void getSensitivity(string[] & msgTypes,string[] & callbacks);
void setSensitivity(string msgType,string callback);
void cancelSensitivity(string msgType);
void asynchronousReception(Message message);
int getNbMessages(void);
Message peekMessage(int pos);
Message getNextMessage(void);
void putBackMessage(Message message);
void onMessage(void);
};
```

Les notions prédéfinies

```
replace randomPicking by 0; //----------------- Controle de l'execution
replace serialPicking by 1;
int getPickingMode(void);
void setPickingMode(int mode);
int getInterval(void);
void setInterval(int interval);
void yield(void); void softYield(void);
int currentExecution(void);
void abort(int execution);
bool isSuspended(int execution);
void suspend(int execution);
void resume(int execution);
void getExecutionStack(Object[] & callers,string[] & modules,
                       int execution);
void printStack(void);
void onExit(string command);
string control(string command,string argument);
```

Les notions prédéfinies

```
int getMaxDepth(void); //---------------------- Controle des recursions
void setMaxDepth(int maxDepth);

int getTime(void); //-------------------------------- Controle du temps
int getClock(void);

bool exist(Object obj); //------------------------ Introspection simple
Object getObject(string objName);
int numberOf(string className);
Object[] getAll(string className);
string[] getClasses(void);

bool loadLibrary(string libName); //--------------- Introduction de code
int parse(string program);
int load(string fileName);
int eval(string code);
```

Les notions prédéfinies

```
int system(string command); //-------------- Interaction avec le systeme
void systemI(string command,string input);
string systemO(string command);
string systemE(string command);
string systemIO(string command,string input);
string systemIE(string command,string input);
string systemOE(string command,string & err);
string systemIOE(string command,string input,string & err);
void systemYield(void);
string systemName(bool detailed);
string getenv(string varName);
void putenv(string varName,string value);
```

Les notions prédéfinies

```
int asciiCode(string char); //------------------------------ Utilitaires
string asciiChar(int code);
string hexaString(int value);
string octalString(int value);
string binaryString(int value);
string locateFile(string fileName);
void showURL(string url,bool newWindow);

string[] commandLine(bool full); //------------------- Entrees / Sorties
void print(Type1 arg1,Type2 arg2,...);
void println(Type1 arg1,Type2 arg2,...);
void warning(Type1 arg1,Type2 arg2,...);
void error(Type1 arg1,Type2 arg2,...);
string format(Type1 arg1,Type2 arg2,...);
string readln(void);
```

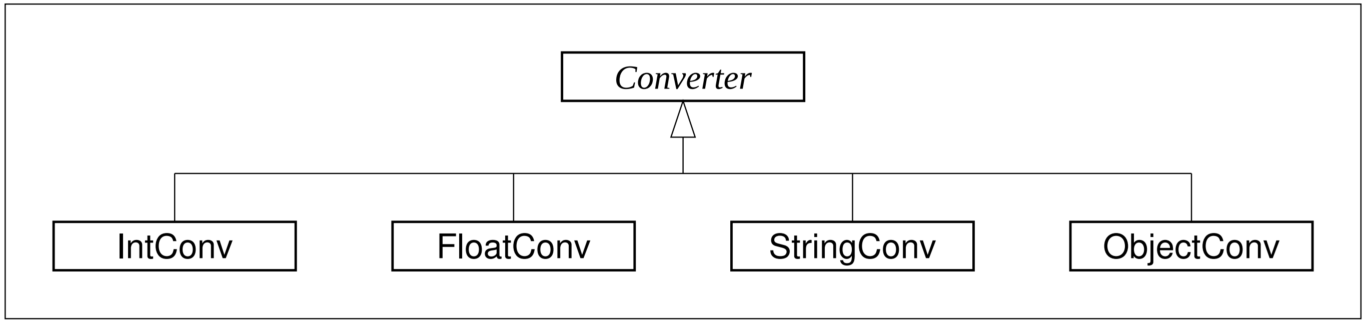
Quelques services utilitaires

▷ **Le paquetage** `control.pkg`

◇ Classe `OrisControl` : communiquer avec la console graphique d'***oRis***

▷ **Le paquetage** `convert.pkg`

◇ Encapsuler et convertir les types de base



▷ **Le paquetage** `maths.pkg`

◇ Fonctions et constantes mathématiques/physiques usuelles

◇ Conversions cartésien $\leftrightarrow$ polaire/sphérique

Quelques services utilitaires

▷ **Le paquetage** `semaphor.pkg`

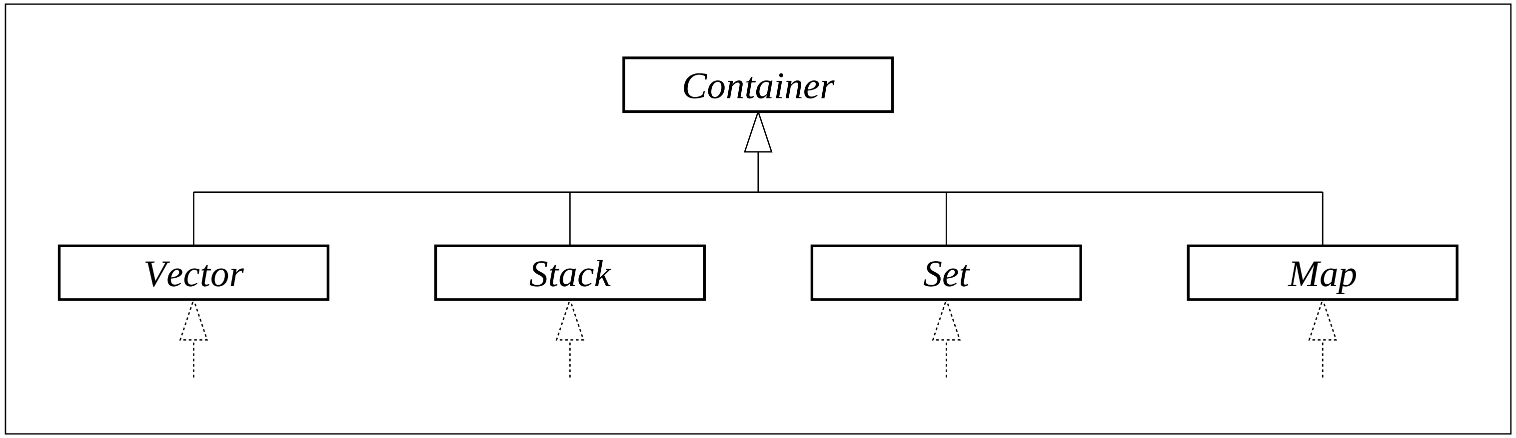
- ◇ Classe `Semaphore` : exclusion mutuelle ***oRis***
- ◇ Relance *FIFO* ou aléatoire

▷ **Le paquetage** `attrCB.pkg`

- ◇ Classe `AttrCB` : liens réflexes
- ◇ Méthodes `before()` et `after()` à surdéfinir

Les conteneurs génériques

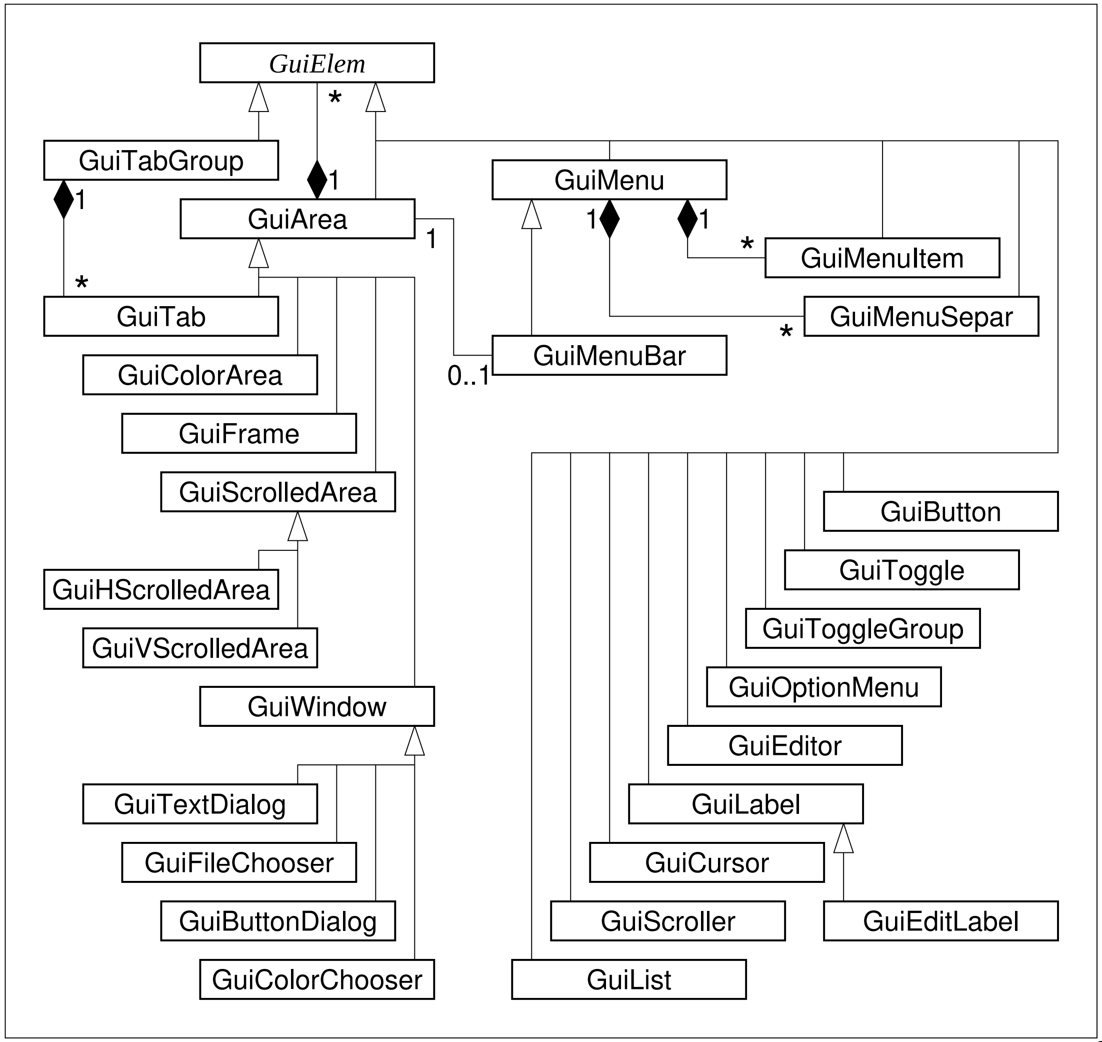
- **Le paquetage** `contain.pkg`
  - Générations de classes de conteneurs génériques **typés**



  - `string defineVectorClass(string elemType,string className);`
  - `string defineStackClass(string elemType,string className);`
  - `string defineSetClass(string elemType,string className);`
  - ```
    string defineMapClass(string keyType,string elemType,
                          string className);
    ```
    ```

Les composants graphiques

▷ **Le paquetage** `gui.pkg`

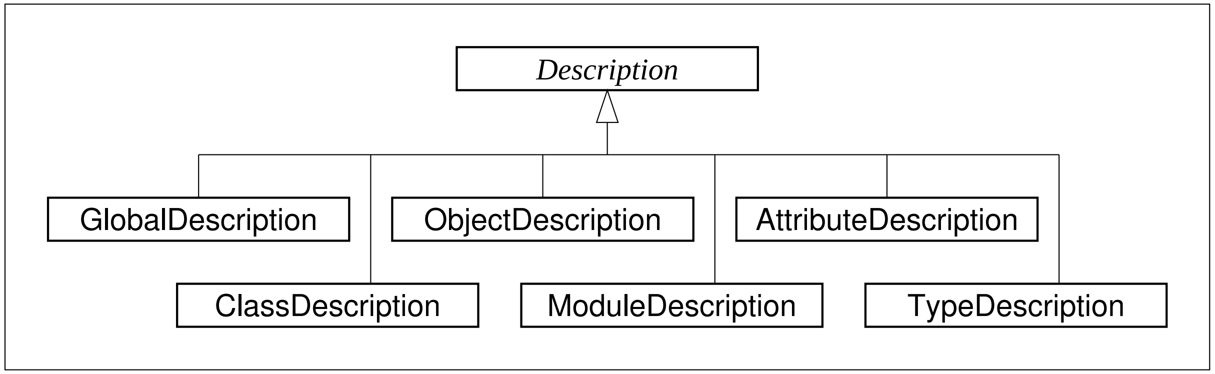


Les composants graphiques

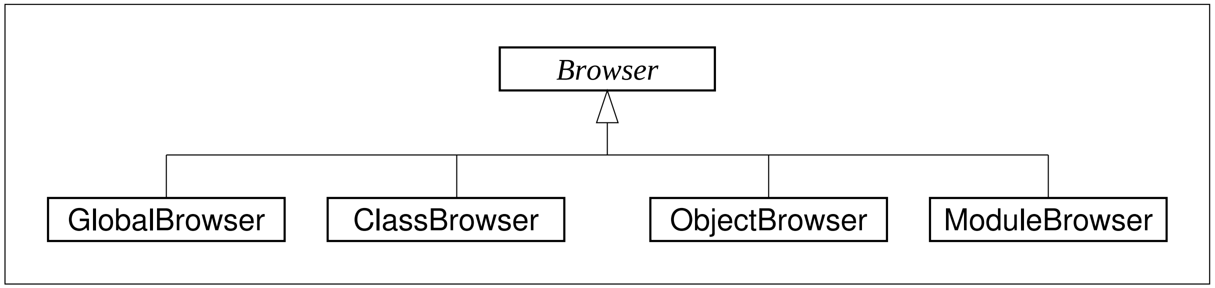
- **Le paquetage** `grapher.pkg`
  - Classes `GuiGrapher` et `Grapher`
  - Traceurs de courbes (continues ou histogrammes)
- **Le paquetage** `object2d.pkg`
  - Classes `Object2d`, `GuiViewer2d` et `Viewer2d`
  - Objets situés dans le plan, formes graphiques simples
  - Placement, perception, lancer de rayon et visualisation
- **Le paquetage** `object3d.pkg`
  - Classes `Object3d`, `Shape3d`, `GuiViewer3d` et `Viewer3d`
  - Objets situés dans l'espace
  - Forme $\equiv$ {points,faces,couleurs,textures}
  - Placement, perception, lancer de rayon et visualisation

Les moyens de réflexion

- **Le paquetage** `reflect.pkg`
  - Classes décrivant toutes les notions de l'application

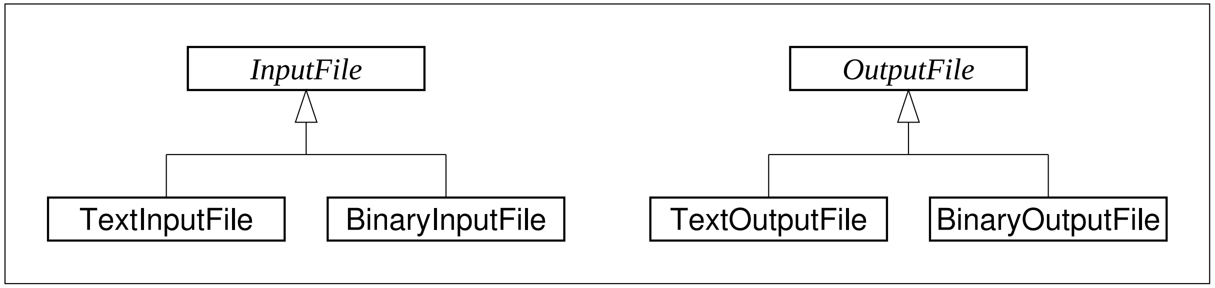


- **Le paquetage** `browser.pkg`
  - Classes d'inspection graphique de l'application



Ouverture à d'autres systèmes

- **Le paquetage** `memory.pkg`
  - Classes `Memory`, `MemoryInput` et `MemoryOutput`
  - Blocs mémoire au *format réseau*
  - Communication par fichier ou par réseau
- **Le paquetage** `files.pkg`
  - Utilisation de fichiers



Ouverture à d'autres systèmes

- **Le paquetage** `tcpUdp.pkg`
  - Classes `TCPListener`, `TCPConnection` et `UDPTransmitter`
  - Conservation des limites des messages en *TCP*
  - Diffusion et point-à-point en *UDP*
  - Communication non-bloquante
- **Le paquetage** `ipc.pkg`
  - Classes `IPCMemory` et `IPCMutex`
  - Communication avec des processus externes à *oRis*

Ouverture à d'autres langages

- **Le paquetage** `prolog.pkg`
  - Invocation d'un interpréteur *Prolog*
  - Depuis *oRis* : `void callProlog(string goal);`
  - Depuis *Prolog* : opérateur `--->`
- **Le paquetage** `java.pkg`
  - Classes `JavaInterface` et `JavaObject`
  - Communication bidirectionnelle avec un processus *Java* externe
  - Utilisation de l'interpréteur *BeanShell*
  - Un objet *Java* $\leftrightarrow$ un `JavaObject` en *oRis*

Perspectives

- **Environnement 3D ouvert** (travail en cours)
  - *VRML* dynamique, scènes, niveaux de détails, intersections ...
- **API *C++* facile d'accès**
  - Permettre aux utilisateurs de réaliser leurs propres *plug-ins*
- **Introduction d'un langage déclaratif**
  - Comportement de plus haut niveau pour les entités autonomes
- **Procédé d'activation de plus haut niveau**
  - Gestion du temps physique
- **Distribution** (travail en cours)
  - *Dead-reckoning*, *HLA*, messages distribués ...

Table des matières

Table des matières