MOOC semaine 6
Héritage multiple

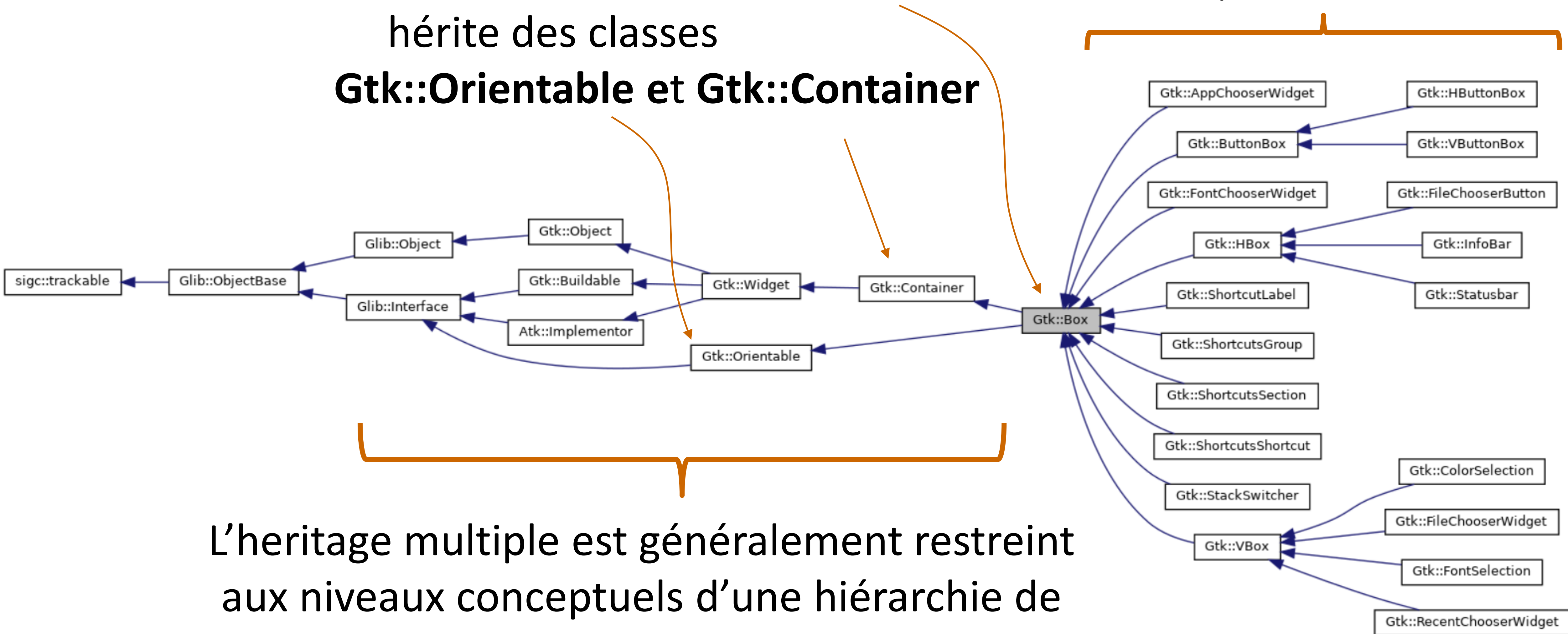
Objectif: Définir clairement des concepts (ou propriétés) *indépendants* dans des classes et s'en servir comme un menu pour créer des classes plus élaborées par héritage multiple.

Plan:

- Brève illustration sur Gtk
- Le lien virtuel: un second usage pour le mot clef **virtual**
- Exemples: configuration en losange avec/sans lien virtuels

Exemple: la classe **Gtk::Box** hérite des classes **Gtk::Orientable et Gtk::Container**

Spécialisation



L'heritage multiple est généralement restreint aux niveaux conceptuels d'une hiérarchie de classe et pratiqué sur très peu de classes

Spécialisation

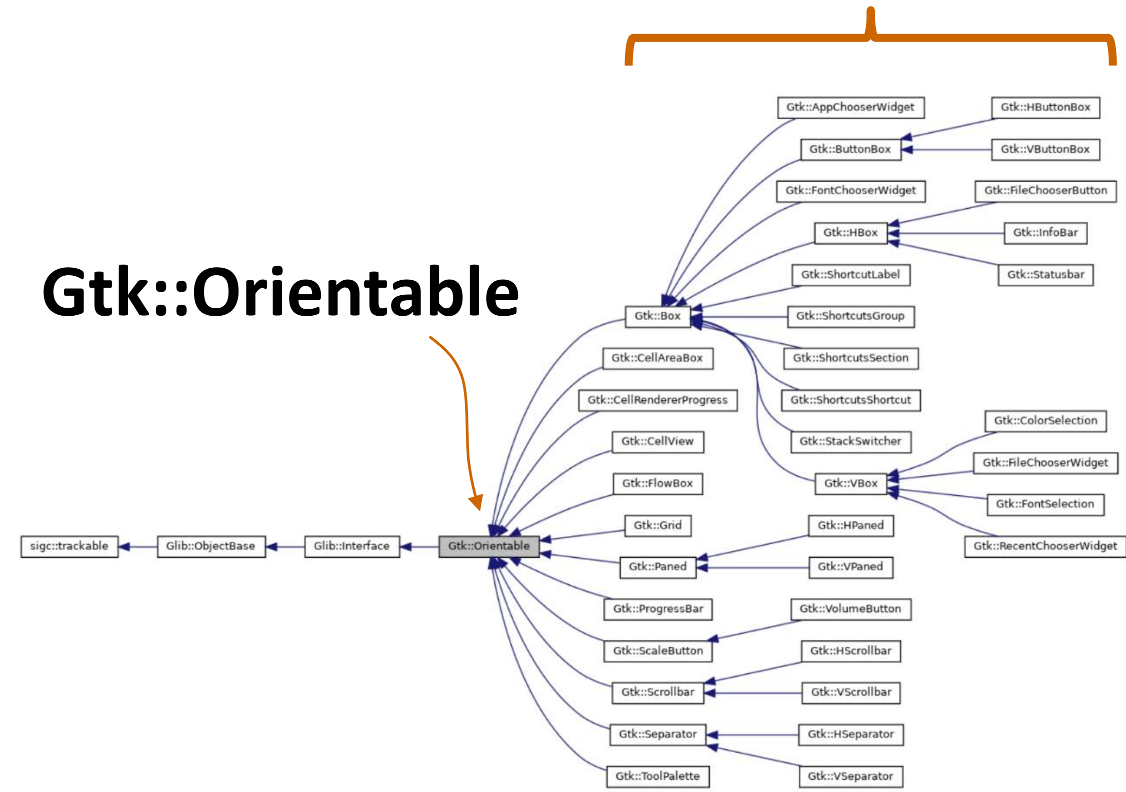
Gtk::Container

sigc::trackable
Glib::ObjectBase
Glib::Object
Glib::Interface
Gtk::Object
Gtk::Buildable
Atk::Implementor
Gtk::Widget
Gtk::Container

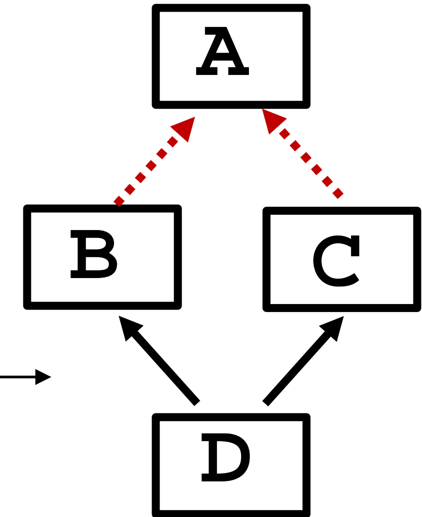
Gtk::Bin
Gtk::Box
Gtk::Fixed
Gtk::FlowBox
Gtk::Grid
Gtk::HeaderBar
Gtk::IconView
Gtk::Layout
Gtk::ListBox
Gtk::MenuShell
Gtk::Notebook
Gtk::Paned
Gtk::Socket
Gtk::Stack
Gtk::Table
Gtk::TextView
Gtk::Toolbar
Gtk::ToolItemGroup
Gtk::ToolPalette
Gtk::TreeView

Gtk::ActionBar
Gtk::Alignment
Gtk::Button
Gtk::ComboBox
Gtk::EventBox
Gtk::Expander
Gtk::FlowBoxChild
Gtk::Frame
Gtk::HandleBox
Gtk::ListBoxRow
Gtk::MenuItem
Gtk::Overlay
Gtk::Popover
Gtk::Revealer
Gtk::ScrolledWindow
Gtk::SearchBar
Gtk::StackSidebar
Gtk::ToolItem
Gtk::Viewport
Gtk::Window

Spécialisation



Déclaration de l'héritage multiple



Indiquer à la déclaration la liste des classes dont on hérite:

Exemple: `class D : public B, public C { };`

L'indication d'un *lien virtuel* est nécessaire **au niveau supérieur des classes parentes (ici B et C)** pour éviter la duplication des attributs provenant de la classe A :

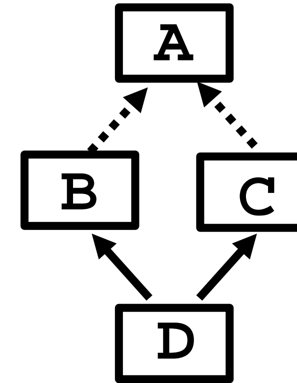
Exemple:

```
class B : public virtual A {};
class C : public virtual A {};
```

Points à surveiller: ordre des classe => ordre des constructeurs, constructeur par défaut

Exemple 1

```
class A
{
public:
  A(int x) : a(x) {}
private:
  int a;
};

class B : public virtual A
{
public:
  B() : A(0) {}
};

class C : public virtual A
{
public:
  C() : A(1) {}
};

class D : public B, public C
{
};

int main()
{
  D d1;
  return 0;
}
```



Cet exemple compile-t-il ?

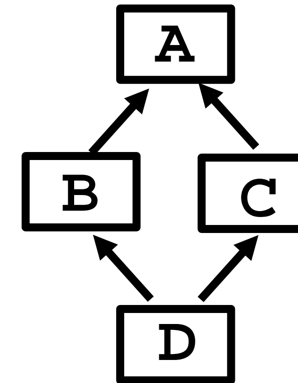
Si non: pourquoi ?

EPFL

Variante

```cpp
class A
{
public:
  A():a(0) {}
  int obsene;
private:
  int a;
};

class B : public A
{
};

class C : public A
{
};

class D : public B, public C
{
};

int main()
{
  D d1;
  cout << d1.B::obsene << endl;
  cout << d1.C::obsene << endl;
}
```

Cet exemple syntaxiquement correct, **sans lien virtuel**, illustre la **duplication de l'attribut** dans la classe D



Affichage possible à l'exécution:

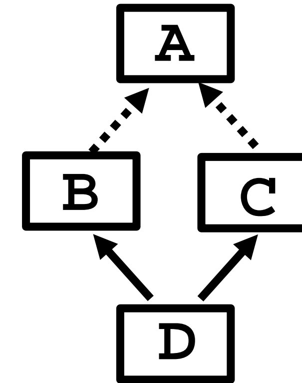
```
4196832
4196240
```

*Rappel: ne **PAS** rendre des attributs public*

Exemple 2

```
class A
{
public:
  void f() const { cout << "Arg!"; }
};

class B : public virtual A
{ };

class C : public virtual A
{ };

class D : public B, public C
{ };

int main()
{
  D d1;
  d1.f();
  return 0;
}
```

Exemple avec **lien virtuel**, illustre le nombre d'appels du constructeur de la classe A



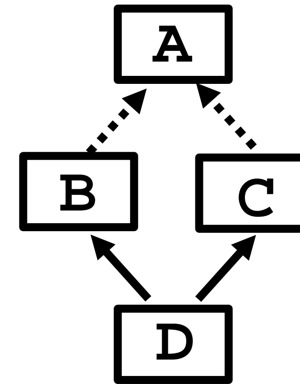
Choisir une réponse:

- Ce code ne compile pas
- Il compile et n'affiche rien
- Il compile et affiche **`Arg!`**
- Il compile et affiche **`Arg!Arg!`**

```
class A
{
public:
  void f() const { cout << "A "; }
};

class B : public virtual A
{
public:
  void f() const { cout << "B "; }
};

class C : public virtual A
{
public:
  void f() const { cout << "C "; }
};

class D : public B, public C
{ };

int main()
{
  D d1;
  d1.f();
}
```

Exemple 3

Exemple avec **lien virtuel**



Choisir une réponse:

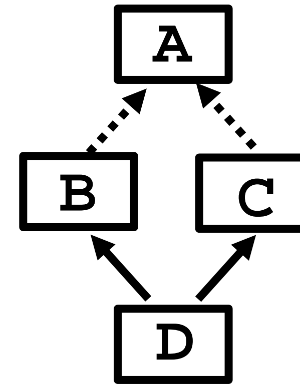
- Ce code ne compile pas
- Il compile et affiche **ABC**
- Il compile et affiche **AB**
- Il compile et affiche **AC**
- Il compile et affiche **BC**
- Il compile et affiche **A**

EPFL

```
class A
{
public:
  virtual void f() const { cout << "A "; }
};

class B : public virtual A
{
public:
  void f() const { cout << "B "; }
};

class C : public virtual A
{
public:
  void f() const { cout << "C "; }
};

class D : public B, public C
{ };

int main()
{
  return 0;
}
```

Exemple 4

Exemple avec **lien virtuel**
et **méthode virtuelle f()**



Choisir une réponse:

- Ce code ne compile pas
- Il compile et n’affiche rien

Elément de réponse: *C++11 Standard 10.3/2 "In a derived class, if a virtual member function of a base class subobject has more than one final overrider the program is ill-formed.“*

EPFL

Points importants

l’héritage multiple est adapté pour la définition de classes de haut niveau à partir de concepts indépendants.

La phase de conception est particulièrement importante pour pouvoir étendre la hiérarchie de classe à la fois par dérivation classique (spécialisation) mais aussi en ajoutant des concepts supplémentaires.

Les fonctionalités du C++ sont loin d’être épuisées mais nous disposons déjà d’un outil puissant du point de vue de l’expression de solutions à la fois conceptuellement élaborées et performantes en pratique.