Université de Reims Champagne-Ardenne
UFR Sciences Exactes et Naturelles
Licence Informatique
INFO0402: Programmation Orientée Objet

TD N°7

Exercice 4: Pointeurs de fonction vs Héritage

On veut donner un exemple concret qui permet de comparer les deux approches en terme de flexibilité.

1. soit un objet de type `bind` pour lequel il existe deux grandes familles (`standard`, `safe`) d'implémentation de deux méthodes `open`, `close`. Expliquer comment obtenir organiser les classes afin d'obtenir les implémentations choisies.
2. en cours de connexion, il devient possible de passer de standard à safe et vice-versa. Quel problème ce changement pose-t-il é l'approche sous forme de classe ?
3. proposer une alternative avec des classes.
4. proposer une implémentation utilisant les pointeurs de fonction.
5. quel est un autre avantage de l'implémentation par pointeur ?

Solution:

1. voir le code :

```
struct iConnect {
  virtual void open() = 0;
  virtual void close() = 0;
};
```

```
class Bind : iConnect {
  // partie commune
};
```

```
class StandardBind : public Bind {
public:
  void open() override { /* open for standard bind */ }
  void close() override { /* close for standard bind */ }
};
```

```
class SafeBind : public Bind {
public:
  void open() override { /* open for safe bind */ }
  void close() override { /* close for safe bind */ }
};
```

```
Bind b = new StandardBind();
b->open();
...
b->close();
```

2. suppose que l'interface peut être changée après création de l'objet. Par exemple de la manière suivante :

```
struct iConnect {
  virtual void open() = 0;
  virtual void close() = 0;
};
```

```
class StandardBind : public iConnect {
public:
  void open() override { /* open for standard bind */ }
  void close() override { /* close for standard bind */ }
};
```

```
class SafeBind : public iConnect {
public:
  void open() override { /* open for safe bind */ }
  void close() override { /* close for safe bind */ }
};
```

```
class Bind : iConnect {
public: enum Mode { standard, safe };
private:
  iConnect *connect;
  Mode      mode;
```

```
public:
  Bind(Mode m) : mode(m) {
    if (mode == standard)  connect = new StandardBind;
    else if (mode == safe) connect = new SafeBind;
  }
```

```
  void open() { connect->open(); }
  void close() { connect->close(); }
```

```
  void SwitchToSafe() {
    if (mode != safe) {
      delete connect;
      connect = new SafeBind;
      mode = safe;
    }
  }
  void SwitchToStandard() {
    if (mode != standard) {
      delete connect;
      connect = new StandardBind;
      mode = standard;
    }
  }
};
```

3. Avec le polymorphisme, le type d'un objet défini la totalité de ses fonctions. Il faudrait donc changer le type de l'objet en cours de l'exécution.

4. On propose d'avoir une classe ne contenant que les implémentations qui est allouée et réallouée en fonction de l'implémentation souhaitée.

```
struct iConnect {
  virtual void open() = 0;
  virtual void close() = 0;
};
```

```
class StandardBind : public iConnect {
public:
  void open() override { /* open for standard bind */ }
  void close() override { /* close for standard bind */ }
};
```

```
class SafeBind : public iConnect {
public:
  void open() override { /* open for safe bind */ }
  void close() override { /* close for safe bind */ }
};
```

```
class Bind : iConnect {
public: enum Mode { standard, safe };
private:
  iConnect *connect;
  Mode      mode;
```

```
public:
  Bind(Mode m) : mode(m) {
    if (mode == standard)  connect = new StandardBind;
    else if (mode == safe) connect = new SafeBind;
  }
```

```
  void open() { connect->open(); }
  void close() { connect->close(); }
```

```
  void SwitchToSafe() {
    if (mode != safe) {
      delete connect;
      connect = new SafeBind;
      mode = safe;
    }
  }
  void SwitchToStandard() {
    if (mode != standard) {
      delete connect;
      connect = new StandardBind;
      mode = standard;
    }
  }
};
```

5. implémentation avec pointeurs

```
class Bind {
public:
  enum Mode { standard, safe };
  using PtFun = void(*)();
  PtFun open;
  PtFun close;
```

```
private:
  static void standard_open() { cout << "open for standard bind\n"; }
  static void standard_close() { cout << "close for standard bind\n"; }
  static void safe_open() { cout << "open for safe bind\n"; }
  static void safe_close() { cout << "close for safe bind\n"; }
  Mode       mode;
```

```
public:
  Bind(Mode m) {
    if (m == standard)  SwitchToStandard();
    else if (m == safe) SwitchToSafe();
  }
```

```
  void SwitchToSafe() {
    if (mode != safe) {
      open = safe_open;
      close = safe_close;
      mode = safe;
      cout << "switch to safe\n";
    }
    else cout << "already in safe mode\n";
  }
```

```
  void SwitchToStandard() {
    if (mode != standard) {
      open = standard_open;
      close = standard_close;
      mode = standard;
      cout << "switch to standard\n";
    }
    else cout << "already in standard mode\n";
  }
};
```

```
void exec() {
  Bind *b = new Bind(Bind::standard);
  b->open();
  b->SwitchToSafe();
  b->close();
}
```

6. dans le cas de l'implémentation pointeur, chaque pointeur de fonction étant local à un objet, chaque instance de l'objet pourrait choisir une implémentation différente pour chacune de ses méthodes.