

Chapitre 2

Organiser le code d'une application graphique à l'aide du patron de conception MVP (Modèle – Vue – Présentation)

Ressource R2.02 : Développement d'application avec IHM

Qualité logicielle

- ◆ Quelques exemples de qualité du logiciel :
 - Correction : le logiciel répond à des besoins donnés et n'a pas d'erreur
 - Facilité d'utilisation : facilité d'apprendre à utiliser le logiciel
 - Réutilisabilité : capacité de l'ensemble ou de parties du logiciel à être utilisé dans le développement d'autres produits logiciels
 - Lisibilité : effort demandé pour comprendre le code source du logiciel
 - Efficience : utilisation optimale des ressources matérielles et logicielles pour rendre le service demandé
 - Portabilité : facilité de transfert du logiciel vers d'autres plateformes matérielles ou logicielles
 - Modularité : structuration de l'application en petites unités peu couplées entre elles

Qualité logicielle

- ◆ La qualité dépend aussi du **point de vue adopté** :
 - Les critères pertinents pour les utilisateurs finaux
 - Les critères pertinents pour les ingénieurs devant développer et maintenir le logiciel
- ◆ On parle alors de qualité **externe** et de qualité **interne**
- ◆ **Critères de qualité *plutôt* externe**
 - Correction
 - Facilité d'utilisation
 - Efficience
 - Portabilité
- ◆ **Critères de qualité *plutôt* interne**
 - Réutilisabilité,
 - Modularité
 - Lisibilité
 - Efficience

Mesures pour produire un logiciel de qualité

♦ Structuration bien préparée

- Une structure appropriée du code permet de réduire sa complexité
- Structure appropriée = le code peut être découpé en parties qui peuvent quasiment indépendantes : on parle de **faible couplage**
- L'utilisation de l'abstraction (on cache les détails) permet de rendre le code plus facile à appréhender

♦ La réutilisation de composants a des avantages

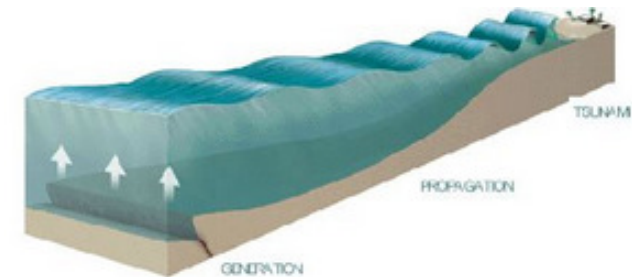
- Les composants logiciels déjà testés et utilisés contiennent moins d'erreurs que ceux composés à partir de zéro
- Les coûts de maintenance et de développement sont réduits et les dépassements évités
- Le développement d'interfaces graphiques (GUI) étant une tâche longue, il faut utiliser des composants réutilisables afin de faciliter l'écriture du logiciel
- Les composants réutilisables sont souvent portables

Notion de patron / pattern

♦ La notion de pattern existe en dehors de l'informatique

— Dans la **nature** :

- Le déferlement des vagues obéit à un pattern (mouvement répétitif), que les mathématiques tentent de modéliser
- Les pétales des fleurs, les fourrures de certains animaux (ex. zèbres) sont créés en obéissant à des patterns (répétition de formes, de couleurs, ...)



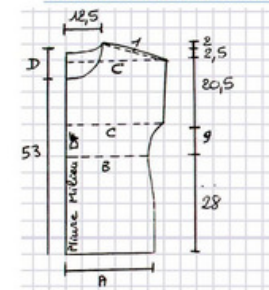
— Patterns **sociaux** :

Formes d'actions sociales qui se répètent et qui ont des probabilités de se reproduire (façon de manger, d'entamer une rencontre avec un autre humain,...)

— Patterns de **production** :

La confection de vêtements utilise des *patrons* (modèles) pour découper les tissus avant de les assembler

T-shirt facile : tailles 36 à 46.



Important : ajouter 5 mm de largeur au milieu dos du patron.

Tissu : 1,50 m x 1,10 m de largeur de jersey

Taille	36	38	40	42	44	46
A	22,5	23,5	24,5	25,5	26,5	27,5
B	21	22	23	24	25	26
C	25	26	27	28	29	30
D	7	7,5	8	8,5	9	9,5

Patron / pattern de conception

◆ En informatique

- Les programmeurs ont tendance à écrire certaines parties de programme par imitation d'autres parties de programmes écrites par d'autres programmeurs plus avancés.
- Cette imitation implique de remarquer le pattern d'un autre code et de l'adapter au programme.
- Le design pattern peut-être vu comme une version forme abstraite de cette activité d'imitation.
- Les design patterns forment un ensemble de règles indiquant comment accompli certaines tâches dans le domaine du développement logiciel.

◆ Définition

Un **patron de conception** est une **solution réutilisable** permettant de répondre à un **type de problème récurrent**.

Utilité - Nature

◆ Utilité des patrons de conception

Ils permettent d'améliorer la qualité logicielle interne des applications

- Modularité
- Flexibilité : facilité d'adaptation en cas de changement
- Réutilisabilité
- Lisibilité

◆ Nature et nombre de patrons de conception

Il existe beaucoup de patrons de conception, chacun spécialisé dans la résolution d'**une problématique** particulière

MVP – Une architecture basée sur 3 composants

- ◆ La patron de conception MVP est un patron destiné aux applications à interface graphique qui préconise l'organisation du code en séparant les **données**, les **traitements** et la **présentation**.

- ◆ **Données → Le Modèle**

C'est le code qui structure les **informations** gérées par l'application, ainsi que les **opérations 'métier'** de l'application.

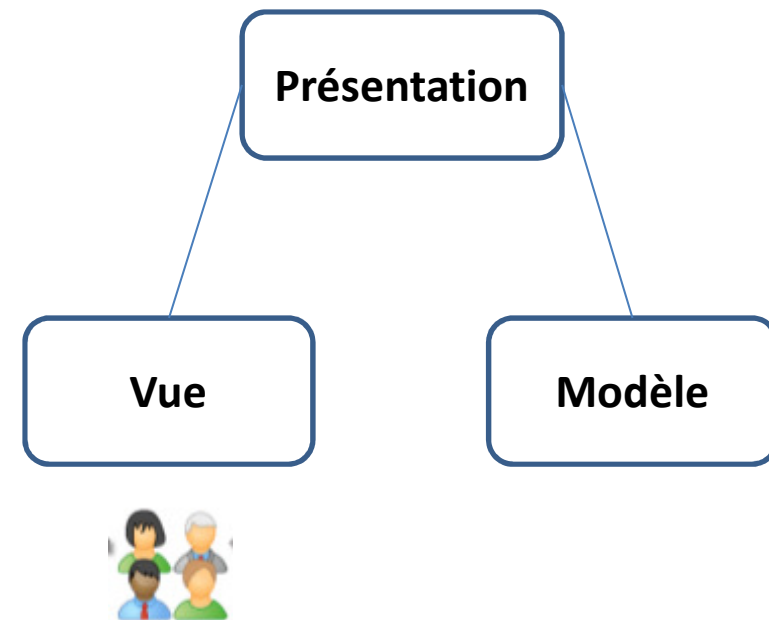
C'est le système d'information sous-jacent à l'application.

- ◆ **Présentation → La Vue**

C'est le code qui **présente** à l'utilisateur les informations du modèle, et qui **interagit** avec l'utilisateur.

- ◆ **Traitements → La Présentation**

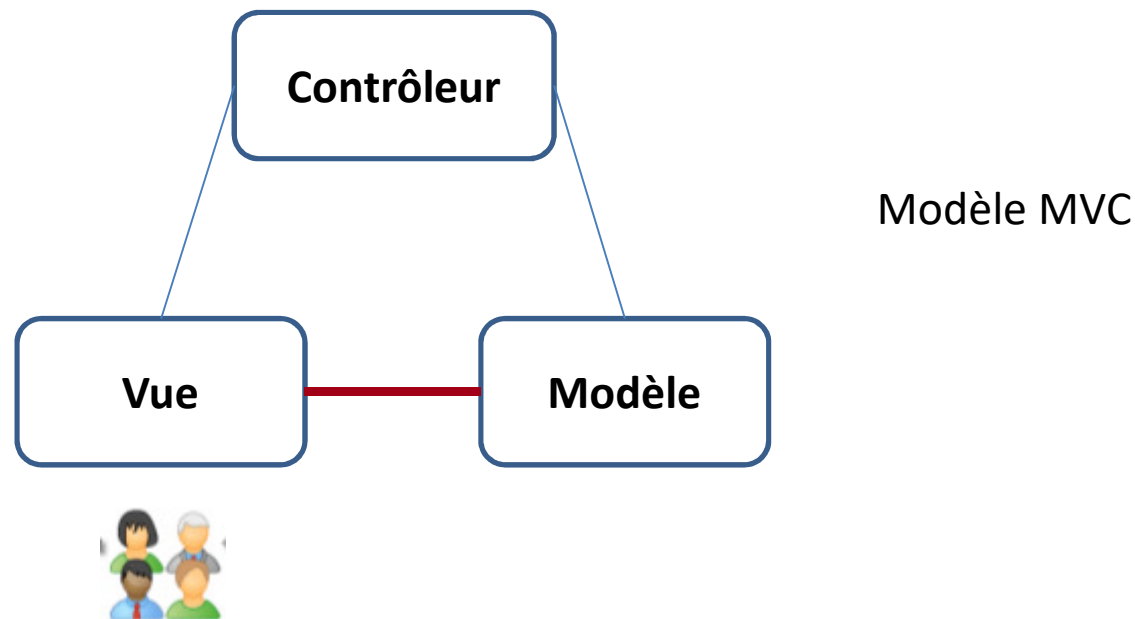
C'est le code qui valide les entrées de l'utilisateur, et qui détermine ce qu'elles signifient pour le modèle.
Tous les échanges entre la Vue et le Modèle passent par la Présentation.



Modèle MVP

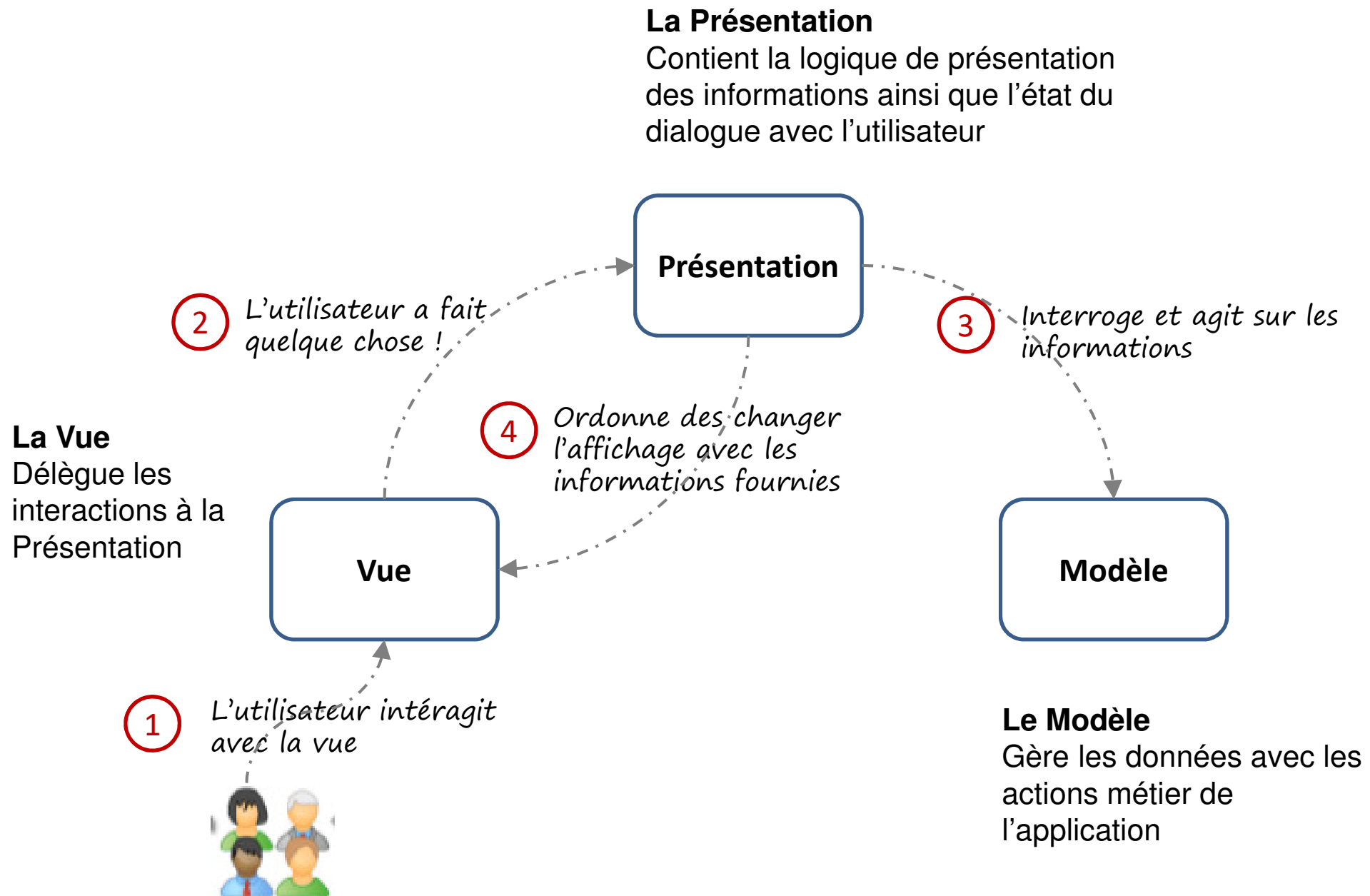
MVP – Un raffinement du modèle MVC

- ♦ La patron de conception MVP est une version améliorée du très célèbre patron Modèle Vue Contrôleur (1979), également destiné aux applications à interface graphique



- ♦ Le modèle MVP réduit le couplage entre la Vue et le Modèle
- ♦ Toutes les interactions entre la Vue et le Modèle passent alors par la Présentation

MVP – Fonctionnement des interactions



MVP – Synthèse des rôles

◆ Le Modèle

- Totalemment centré sur les informations et les actions métier.
- Totalemment déconnecté du dialogue avec l'utilisateur.
- Totalemment indépendant des autres modules de l'application : ne connaît ni la vue ni la présentation.

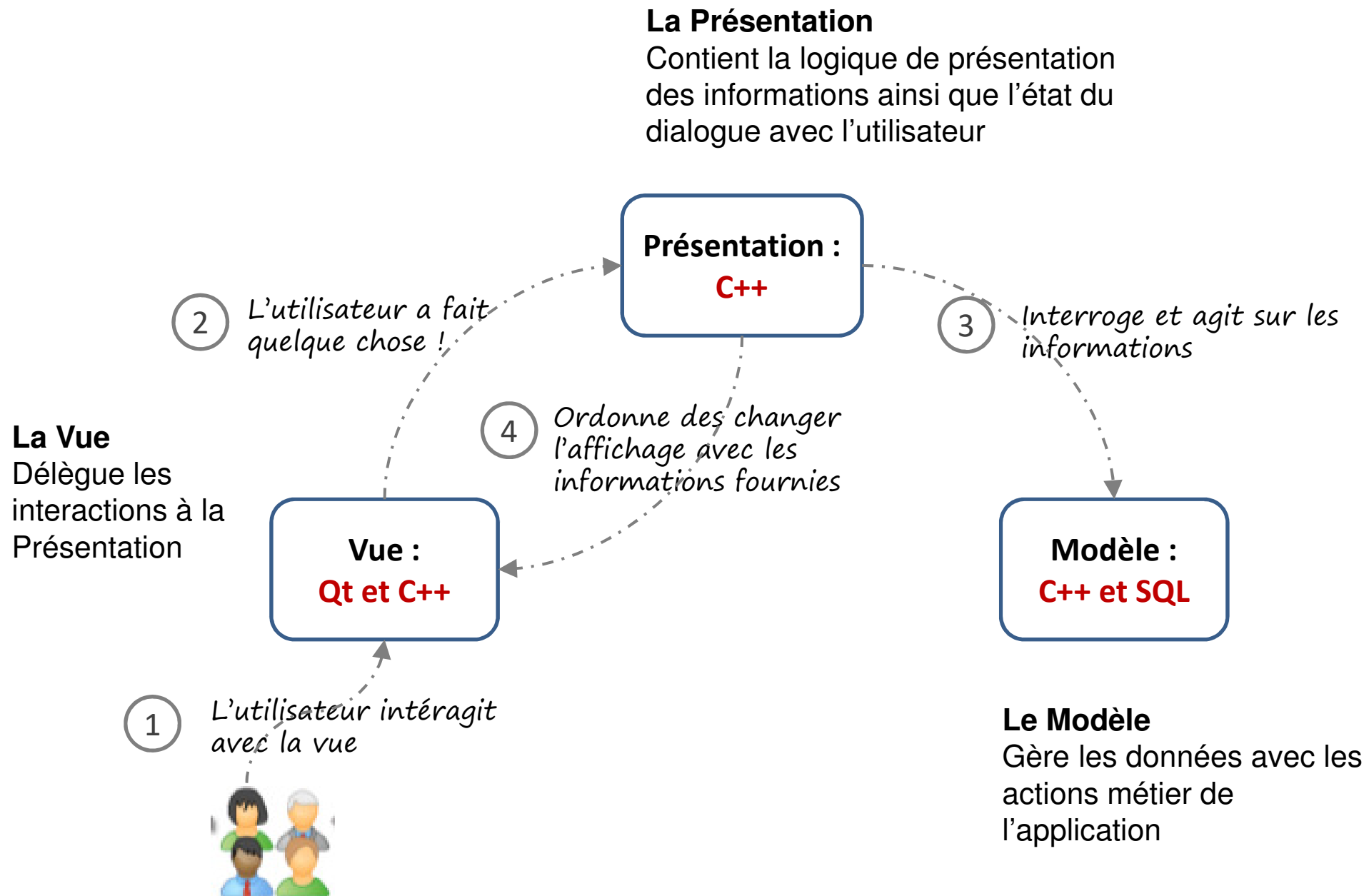
◆ La Vue

- Ne concerne que la gestion graphique.
- Totalemment dépendante de la bibliothèque utilisée.
- A contrario, totalement indépendante du reste de l'application, en particulier, ne connaît rien de la logique de l'application.

◆ La Présentation

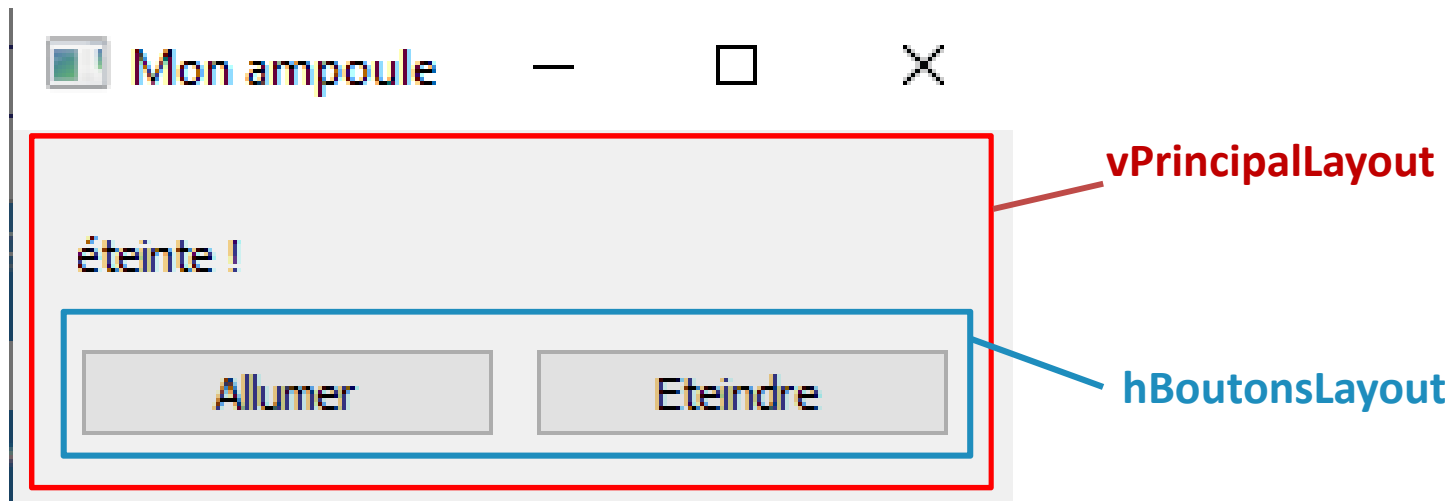
- Contient toute la logique de présentation ainsi que l'état courant du dialogue avec l'utilisateur.
- Elle sait comment réagir aux événements de l'utilisateur, en modifiant les informations du modèle si nécessaire, puis en répercutant les effets sur la vue.
- Totalemment indépendante des classes graphiques utilisées.
- Les classes de la présentation font le lien entre les classes du modèle et les classes de la vue. La présentation est une implémentation du patron Médiateur.

MVP – Technologies d'implémentation en R2.02



Ampoule 1ere version – sans QtDesigner

Éléments d'interface



Lorsque la fenêtre est agrandie, les widgets sont repositionnés/redimensionnés par le layout principal.

Celui-ci s'appuie sur la **stratégie de taille** et d'agencement de chaque widget, définie grâce à la classe **QSizePolicy**.

Par exemple :

- Par défaut, le texte d'un label est aligné à gauche dans le label

Ampoule 1ere version – sans QtDesigner

Une classe Application, une classe graphique

monAmpoule.h

```
1  //... les include etc...
2  class MonAmpoule : public QWidget
3  {
4      Q_OBJECT
5
6  public:
7      MonAmpoule(QWidget *parent = nullptr);
8      ~MonAmpoule();
9
10 public slots:
11     void demandeAllumer();
12     void demandeEteindre();
13
14 private:
15     QLabel *labelEtat; // éteinte / allumée
16     QPushButton *bAllumer;
17     QPushButton *bEteindre;
18
19 };
20 #endif // MONAMPOULE_H
```

Ampoule 1ere version – sans QtDesigner

Une classe Application, une classe graphique

monampoule.cpp (constructeur)

```
1  MonAmpoule::MonAmpoule(QWidget *parent)
2      : QWidget(parent)
3  {
4      // création et paramétrage des objets
5      labelEtat = new QLabel(tr("éteinte !"));
6      bAllumer = new QPushButton(tr("Allumer"));
7      bEteindre = new QPushButton(tr("Eteindre"));
8      setWindowTitle(tr("Ampoule"));
9
10     // Création des layouts
11     QVBoxLayout *vPrincipallayout = new QVBoxLayout;
12     QHBoxLayout *hBoutonsLayout = new QHBoxLayout;
13
14     // Peuplement des layouts
15     hBoutonsLayout->addWidget(bAllumer);
16     hBoutonsLayout->addWidget(bEteindre);
17     vPrincipallayout->addWidget(labelEtat);
18     vPrincipallayout->addLayout(hBoutonsLayout);
19
20     // Associer le layout principal à la fenêtre principale
21     this->setLayout(vPrincipallayout);
22     //...
}
```

Ampoule 1ere version – sans QtDesigner

Une classe Application, une classe graphique

monampoule.cpp (méthodes)

```
1 void MonAmpoule::demandeAllumer()
2 {
3     labelEtat->setText("allumé !");
4     bAllumer->setEnabled(false);
5     bEteindre->setEnabled(true);
6 }
7
8 void MonAmpoule::demandeEteindre()
9 {
10    labelEtat->setText("éteint !");
11    bAllumer->setEnabled(true);
12    bEteindre->setEnabled(false);
13 }
```


Ampoule 2eme version – avec QtDesigner

Une classe Application, une classe graphique

monAmpoule.h

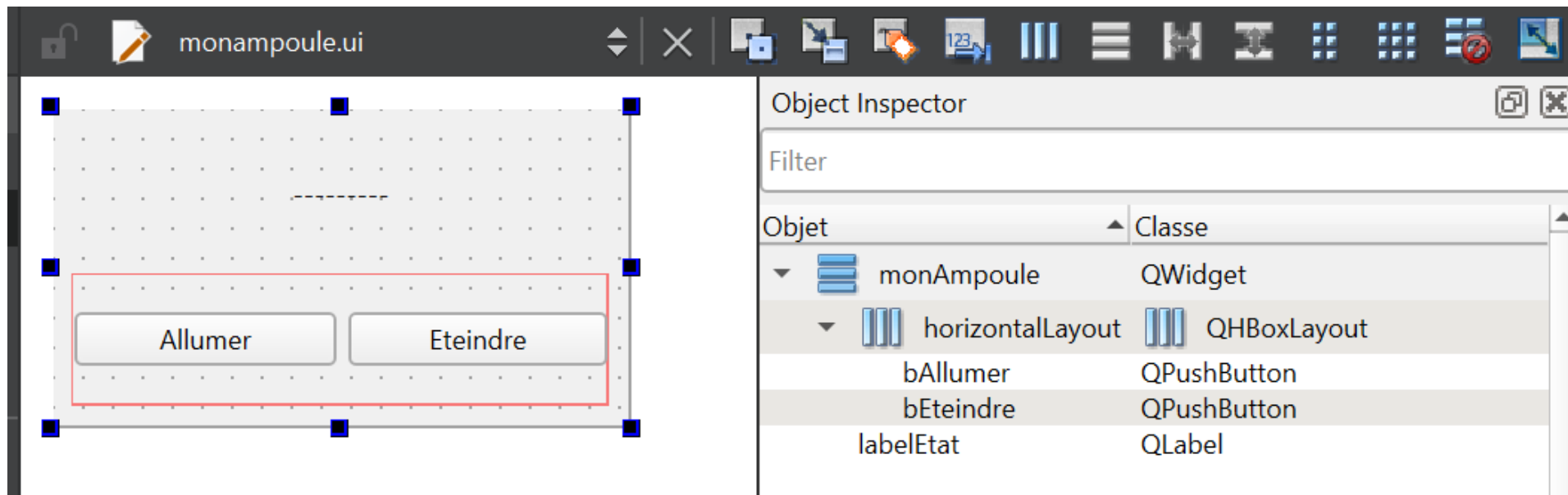
```
1  //... les include etc...
2  class MonAmpoule : public QWidget
3  {
4      Q_OBJECT
5
6  public:
7      MonAmpoule(QWidget *parent = nullptr);
8      ~MonAmpoule();
9
10 public slots:
11     void demandeAllumer();
12     void demandeEteindre();
13
14 private:
15     QLabel *labelEtat; // éteinte / allumée
16     QPushButton *bAllumer;
17     QPushButton *bEteindre;
18
19 };
20 #endif // MONAMPOULE_H
```

Ampoule 2eme version – avec QtDesigner

Le dossier du projet

- ampoule.pro
- main.cpp
- monampoule.cpp
- monampoule.h
- monampoule.ui

Un fichier .ui contenant le dessin de l'interface



Ampoule 2eme version – avec QtDesigner

2 classes, dont 1 graphique, pointant vers les objets d'interface

monampoule.h

```
1  #ifndef MONAMPOULE_H
2  #define MONAMPOULE_H
3  #include <QWidget>
4
5  QT_BEGIN_NAMESPACE
6  namespace Ui { class monAmpoule; }
7  QT_END_NAMESPACE
8
9  class monAmpoule : public QWidget
10 {
11     Q_OBJECT
12
13 public:
14     monAmpoule(QWidget *parent = nullptr);
15     ~monAmpoule();
16 public slots:
17     void demandeAllumer();    // de this
18     void demandeEteindre();  // de this
19 private:
20     Ui::monAmpoule *ui;
21 };
22 #endif // MONAMPOULE_H
```

Ampoule 2eme version – avec QtDesigner

2 classes, dont 1 graphique, pointant vers les objets d'interface

monampoule.cpp (constructeur)

```
1  #include "monampoule.h"
2  #include "ui_monampoule.h"
3
4  monAmpoule::monAmpoule(QWidget *parent)
5      : QWidget(parent)
6      , ui(new Ui::monAmpoule)
7  {
8      ui->setupUi(this);
9
10     // paramétrage initiaux des widgets
11     ui->labelEtat->setText("éteint !");
12     ui->bAllumer->setEnabled(true);
13     ui->bEteindre->setEnabled(false);
14
15     // connexions
16     connect(ui->bAllumer, SIGNAL(clicked()), this, SLOT(demandeAllumer()));
17     connect(ui->bEteindre, SIGNAL(clicked()), this, SLOT(demandeEteindre()));
18 }
...

```

Ampoule 2eme version – avec QtDesigner

2 classes, dont 1 graphique, pointant vers les objets d'interface

monampoule.cpp (méthodes)










```
1  monAmpoule::~~monAmpoule()
2  {
3      delete ui;
4  }
5
6  void monAmpoule::demandeAllumer()
7  {
8      ui->labelEtat->setText("allumé !");
9      ui->bAllumer->setEnabled(false);
10     ui->bEteindre->setEnabled(true);
11
12 }
13
14 void monAmpoule::demandeEteindre()
15 {
16     ui->labelEtat->setText("éteint !");
17     ui->bAllumer->setEnabled(true);
18     ui->bEteindre->setEnabled(false);
19 }
```

Ampoule 3eme version – QtDesigner et MVP

♦ Structure de l'application – 4 classes

- Classe Modele : le modèle
- Classe Presentation : la présentation
- Classe AmpouleVue : la vue
- Classe Application : le processus

♦ Le dossier du projet

-  ampoule.pro
-  ampoulevue.cpp
-  ampoulevue.h
-  ampoulevue.ui
-  main.cpp
-  modele.cpp
-  modele.h
-  presentation.cpp
-  presentation.h

Ampoule 3eme version – QtDesigner et MVP

Le modèle

modele.h

```
1  #ifndef MODELE_H
2  #define MODELE_H
3
4  #include <QObject>
5
6  class Modele : public QObject
7  {
8      Q_OBJECT
9  public:
10     enum UnEtat {eteint, allume};
11     explicit Modele(UnEtat e=eteint, QObject *parent = nullptr);
12     void allumer();
13     void eteindre();
14     UnEtat getEtat();
15 private:
16     UnEtat _etat;
17 };
18
19 #endif // MODELE_H
```

Ampoule 3eme version – QtDesigner et MVP

Le modèle

modele.h

```
1  #include "modele.h"
2
3  Modele::Modele(UnEtat e, QObject *parent)
4      : QObject{parent}, _etat(e)
5  {
6  }
7
8  void Modele::allumer()
9  {
10     _etat = UnEtat::allume;
11 }
12
13 void Modele::eteindre()
14 {
15     _etat = UnEtat::eteint;
16 }
17
18 Modele::UnEtat Modele::getEtat()
19 {
20     return _etat;
21 }
```


Ampoule 3eme version – QtDesigner et MVP

La présentation

presentation.h

```

1  ...
2  #include <QObject>                                // classe non graphique
3  #include "modele.h"
4  class AmpouleVue;
5  class Presentation : public QObject
6  { Q_OBJECT
7  public:
8      explicit Presentation(Modele *m, QObject *parent = nullptr);
9  public:
10     Modele* getModele();
11     AmpouleVue* getVue();
12     void setModele(Modele *m);
13     void setVue(AmpouleVue *v);
14 public slots:                                     // déclenchés par la vue
15     void demandeAllumer();
16     void demandeEteindre();
17 private:
18     Modele *_leModele;                             // la présentation connaît le modèle
19     AmpouleVue *_laVue;                             // et la vue !
20 };
21 #endif // PRESENTATION_H

```

Ampoule 3eme version – QtDesigner et MVP

La présentation

presentation.cpp

```
1  #include "presentation.h"
2  #include "ampoulevue.h"
3  #include <QDebug>
4
5  Presentation::Presentation(Modele *m, QObject *parent)
6      : QObject{parent}, _leModele(m)
7  {
8      _leModele->eteindre();
9  }
10
11  Modele *Presentation::getModele()
12  {
13      return _leModele;
14  }
15
16  AmpouleVue *Presentation::getView()
17  {
18      return _laVue;
19  }
20
```

Ampoule 3eme version – QtDesigner et MVP

La présentation

presentation.cpp

```
21 void Presentation::setModele(Modele *m)
22 {
23     _leModele = m;
24 }
25
26 void Presentation::setVue(AmpouleVue *v)
27 {
28     _laVue = v;
29 }
30
31 void Presentation::demandeAllumer()           // slot déclenché par la vue
32 {
33     _leModele->allumer();
34     _laVue->majInterface(_leModele->getEtat());
35 }
36
37 void Presentation::demandeEteindre()          // slot déclenché par la vue
38 {
39     _leModele->eteindre();
40     _laVue->majInterface(_leModele->getEtat());
41 }
```

Ampoule 3eme version – QtDesigner et MVP

La Vue

ampouleVue.h

```

1  #include <QWidget>                                // seule classe graphique !
2  #include "modele.h"
3  QT_BEGIN_NAMESPACE
4  namespace Ui { class AmpouleVue; }
5  QT_END_NAMESPACE
6  class AmpouleVue : public QWidget
7  { Q_OBJECT
8  public:
9      AmpouleVue(QWidget *parent = nullptr);
10     ~AmpouleVue();
11 public:
12     // pour créer une connexion avec la présentation
13     void nvlleConnexion(QObject *c);
14     void supprConnexion(QObject *c);
15     // ordres reçus par la présentation, avec des valeurs du modèle
16     void majInterface (Modele::UnEtat e);
17 private:
18     Ui::AmpouleVue *ui;
19 };
20 #endif // AMPOULEVUE_H

```

Ampoule 3eme version – QtDesigner et MVP

La Vue

ampouleVue.cpp

```
1  #include "ampouleVue.h"
2  #include "ui_ampouleVue.h"
3
4  AmpouleVue::AmpouleVue(QWidget *parent)
5      : QWidget(parent)
6      , ui(new Ui::AmpouleVue)
7  {
8      ui->setupUi(this);
9  }
10
11 AmpouleVue::~AmpouleVue()
12 {
13     delete ui;
14 }
15
```

Ampoule – avec QtDesigner

La Vue

ampouleVue.cpp

```
16 void AmpouleVue::nvllerConnexion(QObject *c)
17 {
18     //pour se connecter avec la presentation
19     QObject::connect(ui->bAllumer, SIGNAL(clicked()),
20                     c, SLOT(demandeAllumer()));
21     QObject::connect(ui->bEteindre, SIGNAL(clicked()),
22                     c, SLOT(demandeEteindre()));
23 }
24
25 void AmpouleVue::supprConnexion(QObject *c)
26 {
27     //pour se déconnecter de c
28     QObject::disconnect(ui->bAllumer, SIGNAL(clicked()),
29                        c, SLOT(demandeAllumer()));
30     QObject::disconnect(ui->bEteindre, SIGNAL(clicked()),
31                        c, SLOT(demandeEteindre()));
32 }
33
```

Ampoule – avec QtDesigner

La Vue

ampouleVue.cpp

```
34 void AmpouleVue::majInterface(Modele::UnEtat e)
35 {
36     switch (e) {
37         case Modele::allume :
38             ui->labelEtat->setText("allumé !");
39             ui->bAllumer->setEnabled(false);
40             ui->bEteindre->setEnabled(true);
41             break;
42         case Modele::eteint :
43             ui->labelEtat->setText("éteint !");
44             ui->bAllumer->setEnabled(true);
45             ui->bEteindre->setEnabled(false);
46             break;
47         default: break;
48     }
49 }
```

Ampoule 3eme version – QtDesigner et MVP

♦ Le main

- Il crée la vue
- Il crée modèle
- Il crée la présentation (et initialise ses membres privés `_leModele` et `_laVue`)
- Il crée la connexion entre les signals de la vue et les slots de la présentation
- Il ordonne à la vue de s'afficher en cohérence avec l'état initial du modèle

Ampoule 3eme version – QtDesigner et MVP

main.cpp

```
1  #include "ampoulevue.h" #include "presentation.h" #include "modele.h"
2  #include <QApplication>
3
4  int main(int argc, char *argv[])
5  {
6      QApplication a(argc, argv);
7      // créer le modèle
8      Modele *m = new Modele();
9      // créer la présentation et lui associer le modèle
10     Presentation *p = new Presentation(m);
11     // créer la vue
12     AmpouleVue w;
13     // associer la vue à la présentation
14     p->setVue(&w);
15     // initialiser la vue en conformité avec l'état initial du modèle
16     p->getVue()->majInterface(m->getEtat());
17     // connexion des signaux de la vue avec les slots de la présentation
18     w.nvllleConnexion(p);
19
20     // afficher la vue et démarrer la boucle d'attente des messages
21     w.show();
22     return a.exec();
23 }
```

Merci pour
votre attention