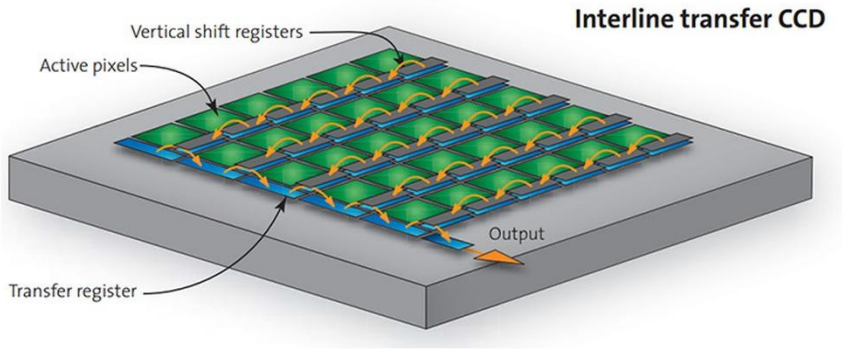
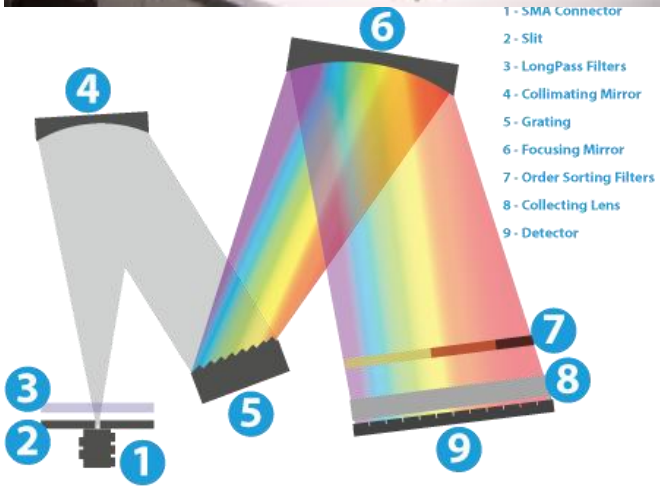
Programowanie obiektowe

24.04.2017

Zadanie

- Software do obsługi spektrometru





Są różne spektrometry

Jak napisać uniwersalny program?

Przykład programu

Program mierzy widmo dla każdej z siatek dyfrakcyjnych

```
int main() {
    ActonSP mono;
    RoperCCD ccd;

    ccd.ustawTrybOdczytu(0);
    ccd.ustawCzasAkumulacji(5);

    for(int i = 0; i < mono.liczbaSiatekDyfrakcyjnych(); i++)
    {
       mono.ustawSiatkeDyfrakcyjna(i);
       vector<int> v = ccd.zmierzWidmo();
       zapisz_widmo(v);
    }
  }
```

Przykład programu

Program mierzy widmo dla każdej z siatek dyfrakcyjnych

```
int main() {
    ActonSP mono; Shamrock mono;
    RoperCCD ccd;

    ccd.ustawTrybOdczytu(0);
    ccd.ustawCzasAkumulacji(5);

    for(int i = 0; i < mono.liczbaSiatekDyfrakcyjnych(); i++)
    {
       mono.ustawSiatkeDyfrakcyjna(i);
       vector<int> v = ccd.zmierzWidmo();
       zapisz_widmo(v);
    }
  }
```

Klasy

```
class Shamrock {
public:
  int liczbaSiatekDyfrakcyjnych()
  {
    return 3;
  }

  void ustawSiatkeDyfrakcyjna(int nr)
  {
  ……
  }
};
```

```
class AndorSP {
public:
  int liczbaSiatekDyfrakcyjnych()
  {
    return 3;
  }

  void ustawSiatkeDyfrakcyjna(int nr)
  {
  ……
  }
};
```

Klasy

```
class RoperCCD {
public:
  vector<int> zmierzWidmo()
  {
    vector<int> v(2000);
    Roper_SDK_run(v.data());
    return v;
  }

  void ustawCzasAkumulacji(double sek)
  {
    Roper_SDK_set_acc_time(sek);
  }

};
```

Klasy

```
class AndorCCD {
public:
  vector<int> zmierzWidmo()
  {
     ????
     ????
     ????
  }

  void ustawCzasAkumulacji(double sek)
  {
     ????
  }

};
```

Inny przypadek:
SDK dostarcza jedynie funkcję
getSpectrum(<czas_akumulacji>)

Klasy

```
class AndorCCD {
public:
  vector<int> zmierzWidmo()
  {


     return getSpectrum(czas);
  }

  void ustawCzasAkumulacji(double sek)
  {
     t = sek;
  }

  const

private:
  int t;
};
```

Inny przypadek:
SDK dostarcza jedynie funkcję getSpectrum(<czas_akumulacji>)

Przykład programu

```
int main() {
     ActonSP mono;
     RoperCCD ccd;

     ccd.ustawTrybOdczytu(0);
     ccd.ustawCzasAkumulacji(5);

     for(int i = 0; i < mono.liczbaSiatekDyfrakcyjna(); i++)
     {
        mono.ustawSiatkeDyfrakcyjna(i);
        vector<int> v = ccd.zmierzWidmo();
        zapisz_widmo(v);
     }
  }
```

Program mierzy widmo dla każdej z siatek dyfrakcyjnych

```
void resetuj(RoperCCD c)
{
   c.ustawTrybOdczytu(0);
   c.ustawCzasAkumulacji(5);
}
```

Nieskuteczne!

```
int main() {
    ActonSP mono;
    RoperCCD ccd;

    resetuj(ccd);

    for(int i = 0; i < mono.liczbaSiatekDyfrakcyjna(); i++)
    {
       mono.ustawSiatkeDyfrakcyjna(i);
       vector<int> v = ccd.zmierzWidmo();
       zapisz_widmo(v);
    }
}
```

```
void resetuj(RoperCCD &c)
{
   c.ustawTrybOdczytu(0);
   c.ustawCzasAkumulacji(5);
}
```

Pracuje na oryginale

```
int main() {
    ActonSP mono;
    RoperCCD ccd;

    resetuj(ccd);

    for(int i = 0; i < mono.liczbaSiatekDyfrakcyjna(); i++)
    {
       mono.ustawSiatkeDyfrakcyjna(i);
       vector<int> v = ccd.zmierzWidmo();
       zapisz_widmo(v);
    }
}
```

Przykład programu

Trzeba napisać funkcję wyznaczającą kalibrację

```
int main() {
     ActonSP mono;
     RoperCCD ccd;

     ccd.ustawTrybOdczytu(0);
     ccd.ustawCzasAkumulacji(5);

     for(int i = 0; i < mono.liczbaSiatekDyfrakcyjna(); i++)
     {
        mono.ustawSiatkeDyfrakcyjna(i);
        vector<int> x = kalibracja(mono, ccd);
        vector<int> y = ccd.zmierzWidmo();
        zapisz_widmo(x,y);
     }
```

Funkcja obliczająca kalibrację

```
vector<int> kalibracja(ActonSP &m, RoperCCD &c)
{
  double rozm = c.rozmiarPixela();
  double gest = m.gestoscSiatki();
  double dl = m.dlugosc();
  <tutaj odpowiedni wzór>
  return ……;
}
```

```
vector<int> kalibracja(ActonSP m, RoperCCD c)
{
  double rozm = c.rozmiarPixela();
  double gest = m.gestoscSiatki();
  double dl = m.dlugosc();
  <tutaj odpowiedni wzór>
  return ……;
}
```

```
vector<int> kalibracja(Shamrock m, AndorCCD c)
{
  double rozm = c.rozmiarPixela();
  double gest = m.gestoscSiatki();
  double dl = m.dlugosc();
  <tutaj odpowiedni wzór>
  return ……;
}
```

```
class Monochromator {
public:
  int liczbaSiatekDyfrakcyjnych() = 0;

  void ustawSiatkeDyfrakcyjna(int nr)
  {
  ……
  }
};
```

Rozwiązanie: dziedziczenie po abstrakcyjnej klasie

```
class Shamrock : public Monochromator {
public:
  int liczbaSiatekDyfrakcyjnych()
  {
    return 3;
  }

  void ustawSiatkeDyfrakcyjna(int nr)
  {
  ……
  }
};
```

Rozwiązanie – dziedziczenie po abstrakcyjnej klasie

```
vector<int> kalibracja(Monochromator &m, CCD &c)
{
  double rozm = c.rozmiarPixela();
  double gest = m.gestoscSiatki();
  double dl = m.dlugosc();
  <tutaj odpowiedni wzór>
  return ……;
}
```

Metody zwykłe vs wirtualne

```
class A {
public:
   void f1() {
     cout << "Wywolana zostala funkcja 1 z klasy A";
   }
   virtual void f2() {
     cout << "Wywolana zostala funkcja 2 z klasy A";
   }
};

class B : public A {
public:
   void f1() {
     cout << "Wywolana zostala funkcja 1 z klasy B";
   }
   void f2() {
     cout << "Wywolana zostala funkcja 2 z klasy B";
   }
};
```

Konstruktor/destruktor

```
class Shamrock : public Monochromator {
public:
  Shamrock() {
  }

  ~Shamrock() {
  }

  int liczbaSiatekDyfrakcyjnych()
  {
    return 3;
  }

  void ustawSiatkeDyfrakcyjna(int nr)
  {
  ……
  }
};
```

prawie zawsze powinien być wirtualny

Kompozycja

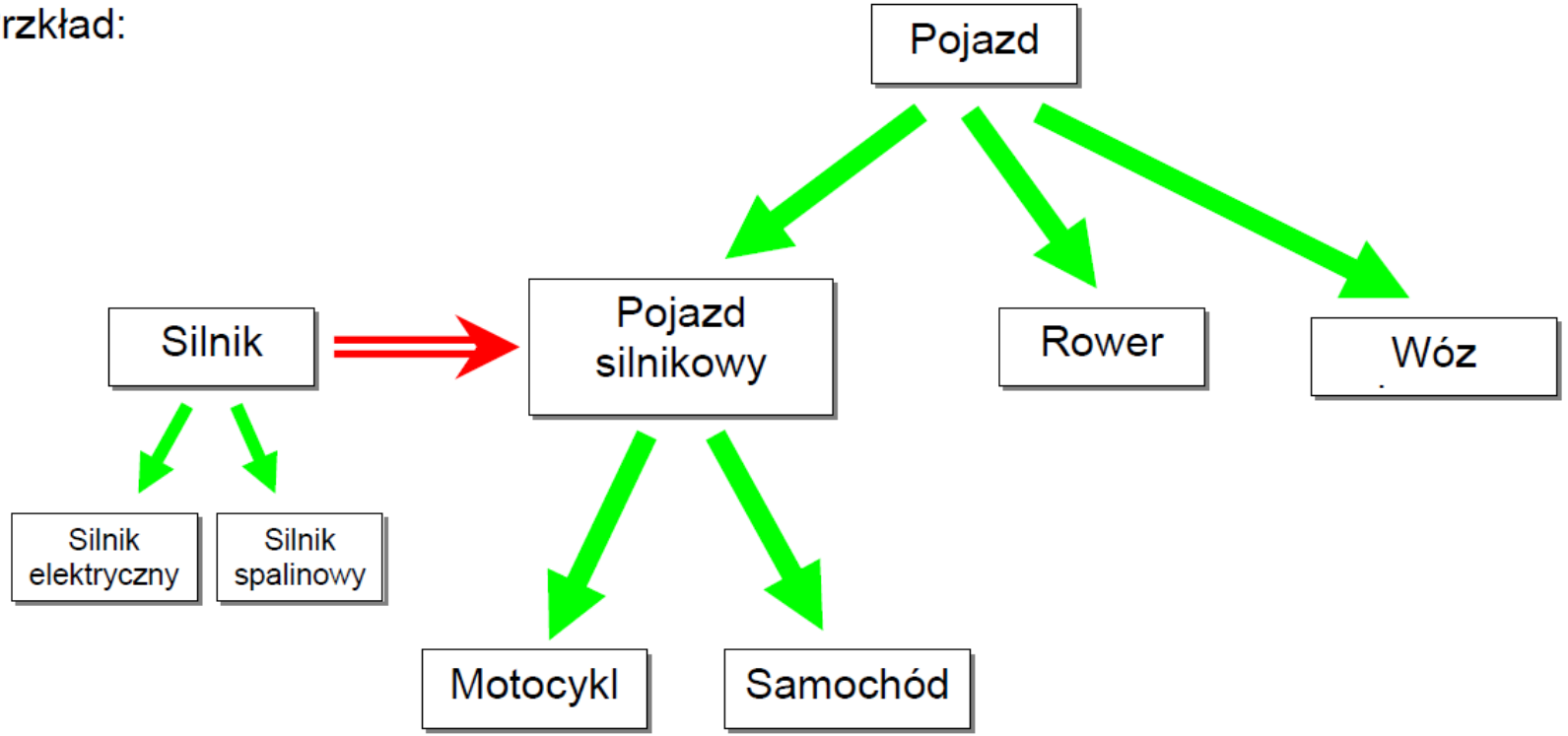
```
class Spektrometr {
public:
  Spektrometr(Monochromator &m, CCD &c) {
    mono = m;
    ccd = c;
  }

  void zmierzWidmoZKalibracja() {
    ……
  }
}

private:
  Monochromator &mono;
  CCD &ccd;
};
```

Związki między klasami: „jest” i „zawiera”

Przkład:



Pojazd silnikowy **jest szczególnym rodzajem** **Pojazdu**

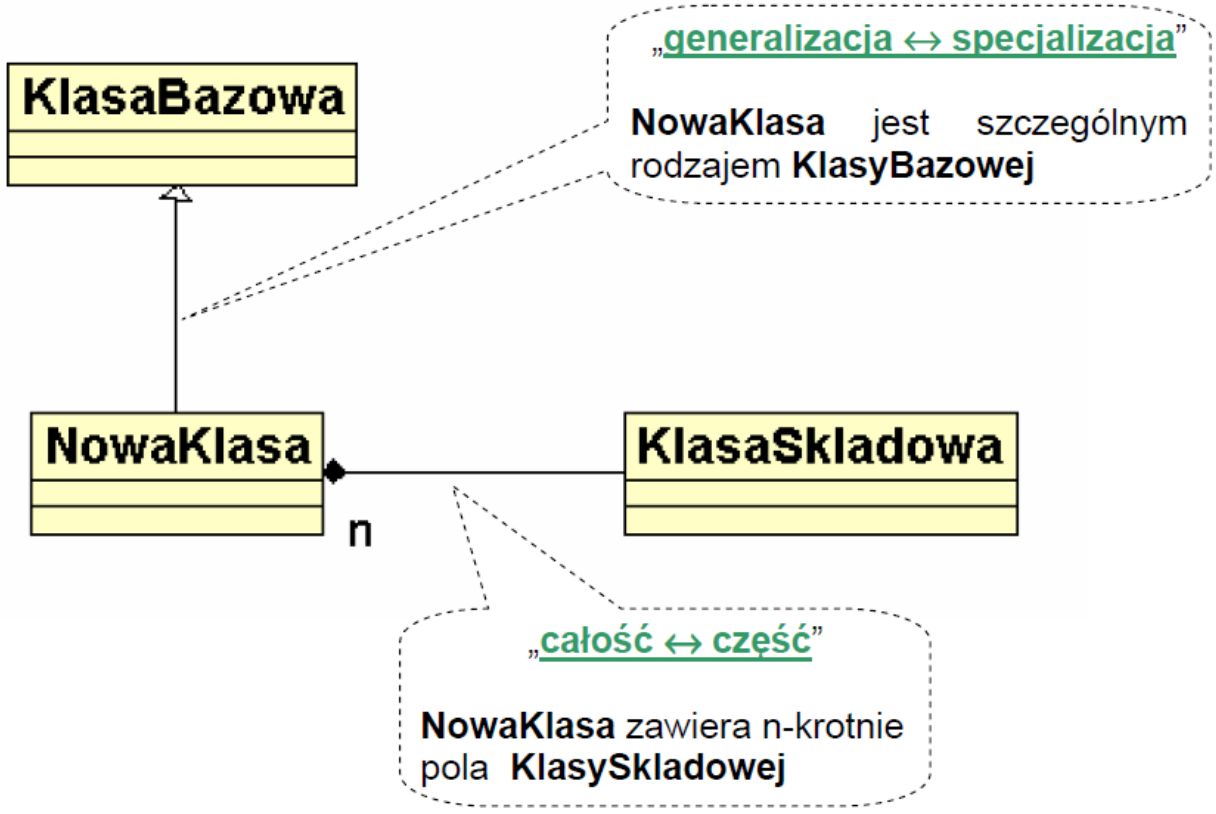
Motocykl **jest szczególnym rodzajem** **Pojazdu silnikowego**

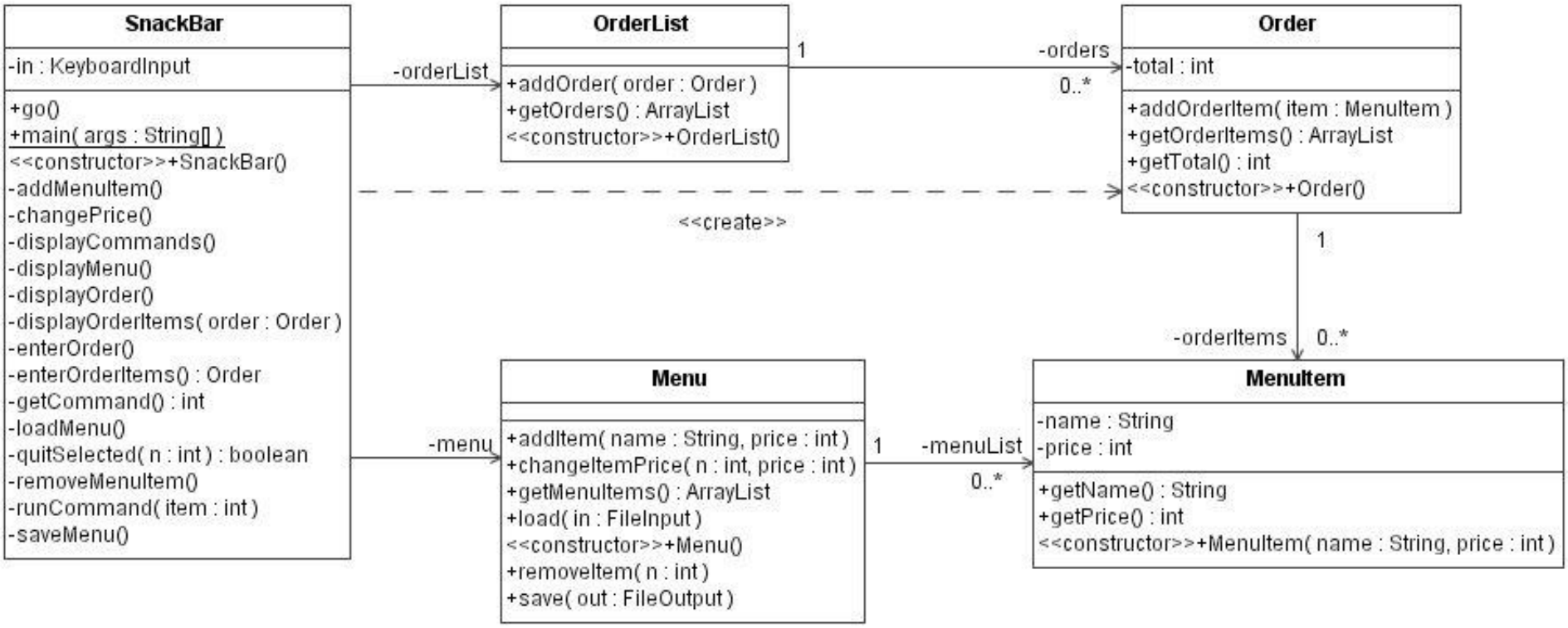
Pojazd silnikowy **zawiera** **Silnik**

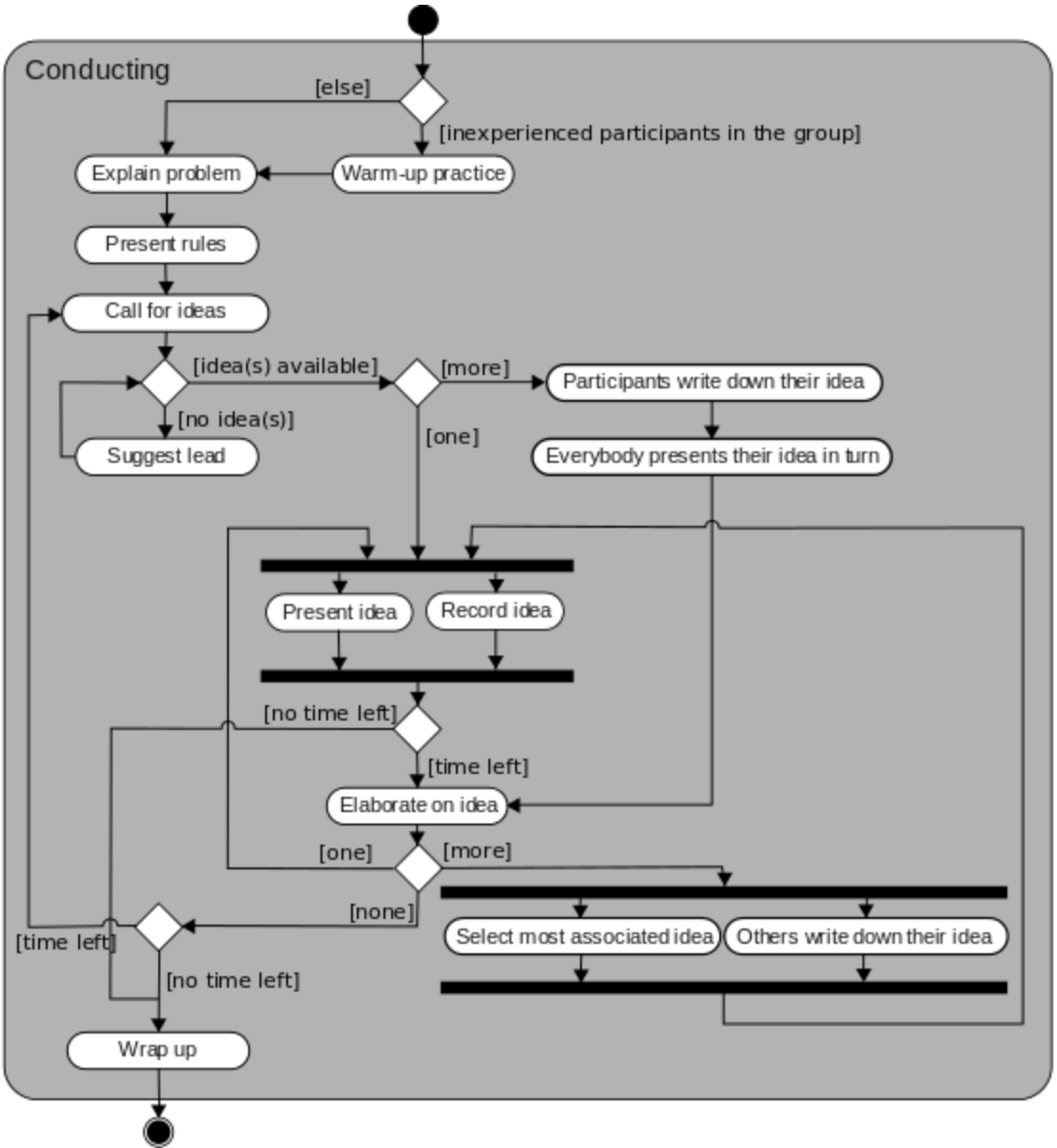
Diagramy klas w języku UML

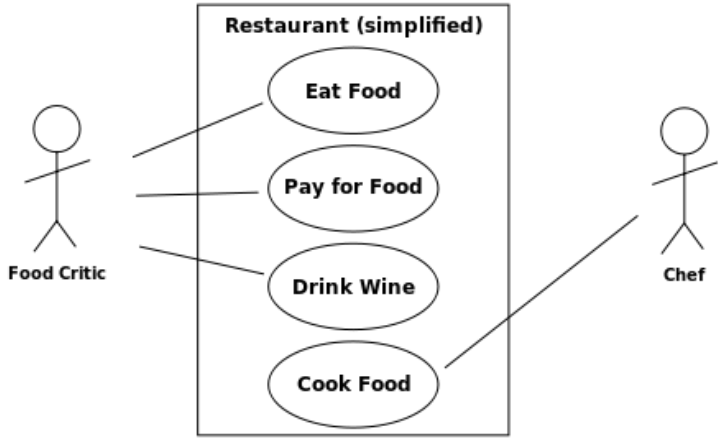
UML (ang. Unified Modeling Language) – zunifikowany język modelowania do tworzenia systemów obiektowo zorientowanych.

Diagram klas pokazuje klasy i zachodzące między nimi relacje.



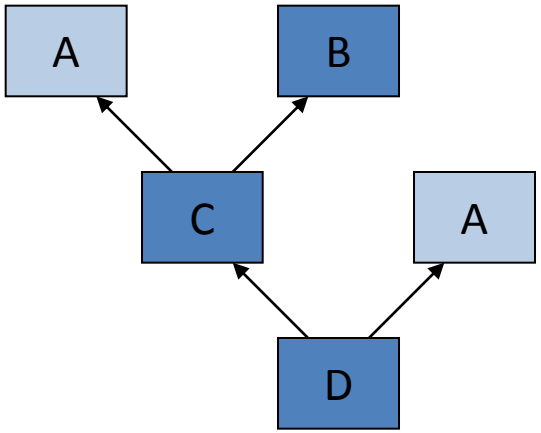

SnackBar
-in : KeyboardInput
+go()
+main(args : String[])
<<constructor>>+SnackBar()
-addMenuItem()
-changePrice()
-displayCommands()
-displayMenu()
-displayOrder()
-displayOrderItems(order : Order)
-enterOrder()
-enterOrderItems() : Order
-getCommand() : int
-loadMenu()
-quitSelected(n : int) : boolean
-removeMenuItem()
-runCommand(item : int)
-saveMenu()
-orderList
OrderList
+addOrder(order : Order)
+getOrders() : ArrayList
<<constructor>>+OrderList()
1
-orders
0..*
Order
-total : int
+addOrderItem(item : MenuItem)
+getOrderItems() : ArrayList
+getTotal() : int
<<constructor>>+Order()
<<create>>
1
-orderItems
0..*
-menu
Menu
+addItem(name : String, price : int)
+changeItemPrice(n : int, price : int)
+getMenuItems() : ArrayList
+load(in : FileInput)
<<constructor>>+Menu()
+removeItem(n : int)
+save(out : FileOutput)
1
-menuList
0..*
MenuItem
-name : String
-price : int
+getName() : String
+getPrice() : int
<<constructor>>+MenuItem(name : String, price : int)

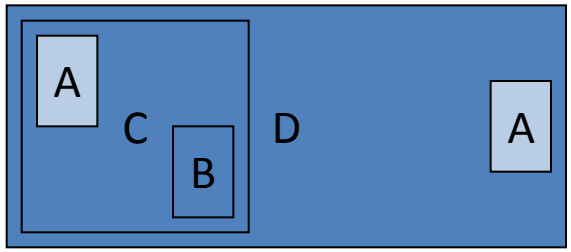

Conducting
[else]
[inexperienced participants in the group]
Explain problem
Warm-up practice
Present rules
Call for ideas
[idea(s) available]
[more]
Participants write down their idea
[no idea(s)]
Suggest lead
[one]
Everybody presents their idea in turn
Present idea
Record idea
[no time left]
[time left]
Elaborate on idea
[one]
[more]
[none]
[time left]
Select most associated idea
Others write down their idea
[no time left]
Wrap up


Restaurant (simplified)
Eat Food
Pay for Food
Drink Wine
Cook Food
Food Critic
Chef

Dziedziczenie wielokrotne (wielodziedziczenie)

- Wielodziedziczenie może prowadzić do wielu skomplikowanych sytuacji: w pojedynczym obiekcie pewna informacja może się wielokrotnie powtórzyć.





Dziedziczenie wirtualne

- Dziedziczenie wirtualne może rozwiązać część problemów z dziedziczeniem wielobazowym.
- Dziedziczenie wirtualne powoduje, że pewne informacje występujące wielokrotnie w obiekcie mogą stać się wspólne dla wielu części.
- Dziedziczenie wirtualne deklaruje się słowem `virtual` występującym na liście pochodzenia przed klasą bazową.
- Konstruktor wirtualnej klasy podstawowej jest wywoływany przed konstruktorami jej klas pochodnych.

Dziedziczenie wirtualne

- Przykład dziedziczenia wirtualnego:

```
class pojazd
    {/*…*/};
class samochód: public virtual pojazd
    {/*…*/};
class łódź: public virtual pojazd
    {/*…*/};
class amfibia: public samochód, public łódź
    {/*…*/};
```