ФАКУЛТЕТ ЗА ЕЛЕКТРОТЕХНИКА И ИНФОРМАЦИСКИ ТЕХНОЛОГИИ

Наследување

– Податочни структури и програмирање–

Наследување (1)

- Еден вид **релација меѓу објектите** воведовме кога **во дефиницијата на една класа** вметнавме **објект од друга класа**
- Но, овде не станува збор за ваков тип на вметнување, туку **идејата е да се дефинира класата со** употреба на **дефиницијата на претходно дефинирана класа**
- **Новата класа ја има функционалноста на постоечката**, но, дополнително и **се додадени функционалности** специфични само за новодефинираната класа

Наследување (2)

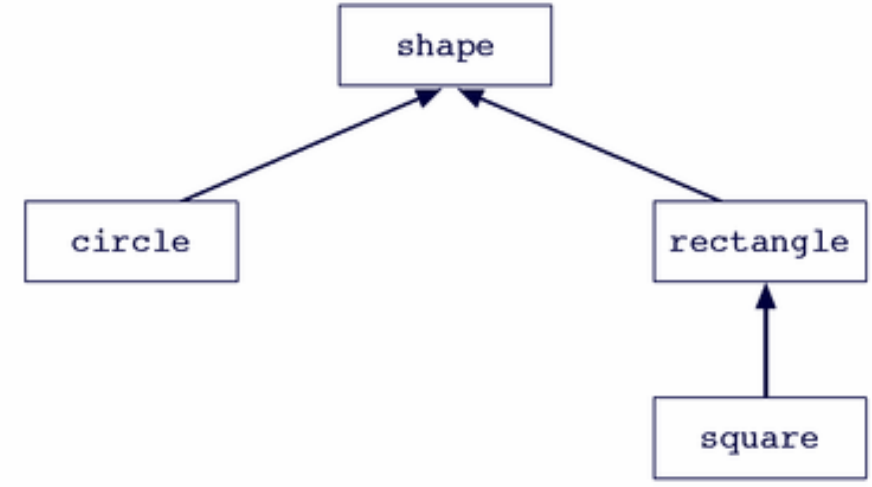
- Наместо за композиција од класи, зборуваме за **изведување на класи**
- **Синоним** за поимот изведување **е терминот наследување**
  - **Новата класа ја наследува функционалноста** на постоечката класа
  - **Објект од постоечката класа не се појавува** како податочен член на новата класа
- Терминолошки
  - **Постоечката класа** се нарекува **основна (базна) класа**
  - **Новата класа** се нарекува **изведена класа**

Наследување (3)

- **Предности** на концептот на наследување
  - **намалено време за развој** на апликацијата,
  - **полесно одржување** на апликацијата, и
  - **лесно проширување** на апликацијата.
- Дизајнерот ги организира **класите хиерархиски**
- За таа цел тој се обидува да ги пронајде сите **генерални или заеднички особини и однесувања** што постојат меѓу класите
- **Погенералните операции** ги лоцира **во класите што се наоѓаат повисоко** во хиерархиското дрво

Наследување (4)

- Дефинира **“is-a” релација**
- **Еднократно наследување**
  - новата класа е изведена **од една основна класа**
- **Повеќекратно наследување**
  - новата класа е изведена **од најмалку две основни класи**



Генерална синтакса на изведена класа

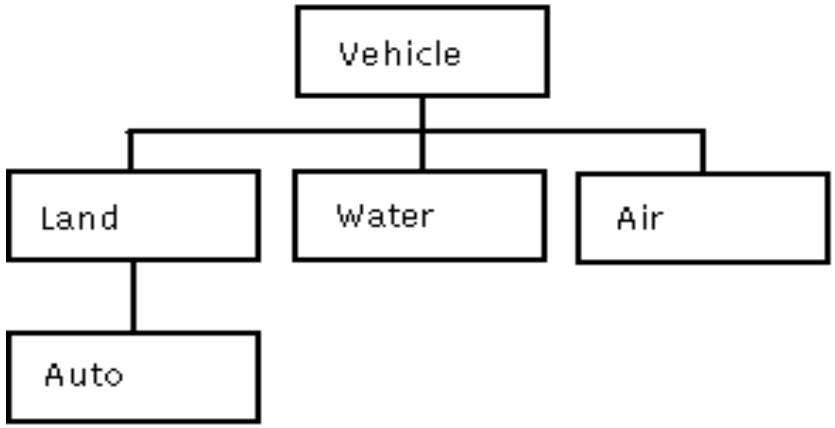
```
class A : base_class_access_specifier B {
      member access specifier(s) :
      ...
            member data and member function(s);
      ...
}
```

- `base_class_access_specifier`
  Валидни вредности се **`public`**, **`private`** и **`protected`**

Како тоа оди во практиката (1)

- **Релација меѓу различни видови објекти** што се однесуваат на **возила** е прикажана на сликата
  - **Автомобилот е специјален случај на земјено** (копнено) **возило**, што пак е специјален случај на возило
  - **Класата `Vozilo` е најопштата** во овој класификациски систем

```
class Vozilo {
public: // konstruktori
        Vozilo();
        Vozilo(int wt);
        // interfejs
        int VratiTezina() const;
        void PostaviTezina(int wt);
private:
        // podatoci
        int tezina;
};
```



Како тоа оди во практиката (2)

- За да се претстават возилата што се движат по земја, потребна е **нова класа** `Zemjeno` **која треба да ја задржи функционалноста на класата** `Vozilo`, **но и да се дополни** со специфичности врзани за возилата што се движат по земја
- Тоа може да го направиме со **композиција на класите** `Vozilo` **и** `Zemjeno`, но композицијата **ќе означува дека сите објекти од класата** `Zemjeno` **содржат возило**, **а не дека објектите** од оваа класа **се само специјален вид на возила**

Како тоа оди во практиката (3)

- Следната дефиниција претставува **композиција од класи**

```
class Zemjeno {
public: void PostaviTezina(int wt);
private: Vozilo v; // sodrzi Vozilo
};
void Zemjeno::PostaviTezina(int wt) { v.PostaviTezina(wt); }
```

- Во примерот, функцијата **`PostaviTezina()`** на класата `Zemjeno` служи само за проследување на аргументи на фунцијата `Vozilo::PostaviTezina()`, што во конкретниот случај не означува дополнителна функционалност, туку само дополнителен код

Наследување во практика (1)

- Јасно е дека **ваквата дефиниција е редундантна**
- Објект на класата `Zemjeno` **е** `Vozilo` и објект од класата `Zemjeno` **не содржи** објект од класата `Vozilo`
- **Оваа релација подобро се опишува со наследување** во кое класата `Zemjeno` се изведува од класата `Vozilo`

```
class Zemjeno : public Vozilo {
public:
   Zemjeno();                          // konstruktori
   Zemjeno(int wt, int sp);            // konstruktori
   void PostaviBrzina(int sp);         // interfejs
   int VratiBrzina() const;            // interfejs
private:
   int brzina;                         // podatoci
};
```

Наследување во практика (2)

- **Класата `Zemjeno` ја содржи цела функционалност на класата `Vozilo` и дополнителни информации** својствени само за неа
- **Примена**:

```
Zemjeno veh(1200, 145);
int main() {
   cout << "Voziloto tezi " << veh.VratiTezina() << endl;
   cout << "Negovata brzina iznesuva " << veh.VratiBrzina() <<endl;
   return (0);
}
```

- **`veh.VratiTezina()` не е директен член на класата `Zemjeno`, но може да се користи** во наредбата **`veh.VratiTezina()`**
  - функцијата е **имплицитен член на класата** бидејќи припаѓа на класата ,,родител” `Vozilo`

Наследување во практика (3)

- Иако класата `Zemjeno` ја содржи функционалноста на класата `Vozilo`, **приватните членови на** класата `Vozilo` **остануваат приватни**
  - до приватните членови на основната класа **може да се пристапи само преку методите** на истата
  - функциите членки на `Zemjeno` мора да ги користат **функциите `VratiTezina()`, `PostaviTezina()` да пристапат до податочниот член `tezina`**; како и која и да е наредба надвор од класата `Vozilo`
  - Ова ограничување е **неопходно за да се зачува принципот на енкапсулација на податоци**
- Класата `Vozilo` може да биде програмски изменета и компајлирана, по што целата програма може да биде повторно линкирана. Самата класа `Zemjeno` може (?) да остане неизменета

Дополнување на класната хиерархија

- Нека дефинираме и класа автомобили `Avtomobil`, која ги претставува специјалниот вид на земјени возила:

```
class Avtomobil : public Zemjeno {
public:
   Avtomobil();                                          // konstruktor
   Avtomobil(int wt, int sp, char const *nm);            // konstruktor
   Avtomobil(Avtomobil const &other);                    // copy konstruktor
   Avtomobil const &operator=(Avtomobil const &other); // dodeluvanje
   ~Avtomobil();                                         // destructor
   char const *VratiIme() const;                         // interfejs
   void PostaviIme(char const *nm);
private:
   char const *ime;                                      // podatoci
};
```

- Класата `Avtomobil` е изведена од класата `Zemjeno`, што пак е изведена од класата `Vozilo`
- `Zemjeno` се нарекува **директна основна класа** на класата `Avtomobil`, а `Vozilo` се нарекува **индиректна основна класа**

Конструктори во изведена класа (1)

- **Изведената класа има сопствени приватни податочни членови**
- **Конструкторите на изведената класа** (директно) **ги иницијализираат само членовите на изведената класа**
- **Извршувањето на конструкторот на изведената класа мора да повика** изведување на **еден од конструкторите на основната класа**
- Ако не е поинаку наведено, **C++** компајлерот генерира **повик до default конструкторот на основната класа** за секој конструктор на изведената класа

Конструктори во изведена класа (2)

```
#include <iostream>
using namespace std;

class BC {
public: BC() { cout << "BC::BC()" << endl; }
private: int x;
};
class DC: public BC {
public: DC() { cout << "DC::DC()" << endl; }
private: int y;
};
int main() {
        DC d;
        return 0;
}
```

```
BC::BC()
DC::DC()
```

Конструктори во изведена класа (3)

```
#include <iostream>
using namespace std;
class BC {
public:          BC() { x = y = -1; }
protected:       int getX() const { return x; }
                 int getY() const { return y; }
private:         int x;
                 int y;
};
class DC : public BC {
public: void write() const { cout << getX()*getY() << endl; }
};
int main() {
        DC d;
//      cout << d.getX() << endl; Ne moze!!!
        d.write();
        return 0;
}
```

1

Конструктори во изведена класа (4)

- **Повиците до другите конструктори во основната класа** се определуваат **во дефиницијата на конструкторите на изведената класа**

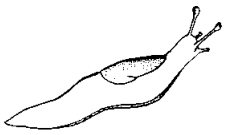
```
class Team {
public:
   Team(int len = 100) { names = new char[maxNo = len]; }
protected:
   char* names;
   int maxNo;
};
class BaseballTeam : public Team {
public:
   BaseballTeam(const char s[], int si) : Team(si) {
        for (int i = 0; i<si; i++) names[i] = s[i];
   }
};
```

Пример, конструктори

```
Animal()
Animal(canary)
Animal(Primate)
Primate()
Animal(Primate)
Primate(4)
Animal(Primate)
Primate(4)
Human()
```

```
#include <iostream>
#include <string>
using namespace std;
class Animal {
public:
        Animal() { cout << "Animal()" << endl; species = "Animal"; }
        Animal(const string s) { cout << "Animal(" << s << ")" << endl; species = s; }
private: string species;
};
class Primate : public Animal {
public:  Primate() : Animal("Primate") { cout << "Primate()" << endl; }
        Primate(int n) : Animal("Primate") {
                cout << "Primate(" << n << ")" << endl; heartCham = n;
        }
private: int heartCham;
};
class Human : public Primate {
public:  Human() : Primate(4) { cout << "Human()" << endl; }
};
int main() {
        Animal slug; //Opisthobranch Molluscs
        Animal tweety("canary");
        Primate godzilla;
        Primate human(4);
        Human jill;
        return 0;
}
```

slug



species

godzilla

species

heartCham

jill

species

heartCham

Деструктори на изведена класа (1)

- **Деструкторите се повикуваат автоматски кога се уништува објект од дадената класа**
- Правилото **важи и за изведените класи**

```
class Osnoven {
public:
        ...          // clenovi
        ~Osnoven(); // destruktor
};
class Izveden : public Osnoven {
public:
        ...          // clenovi
        ~Izveden(); // destruktor
}
... // drug kod
int main() {
        Izveden izveden;
        ...
        return (0);
}
```

- На крајот од `main()` функцијата изведениот објект престанува да постои, и се повикува неговиот деструктор **`Izveden::~Izveden()`**
- Меѓутоа, бидејќи изведениот објект е исто така и основен објект се повикува и деструкторот **`Osnoven::~Osnoven()`**
- Не е потребно да се повика експлицитно **`Osnoven::~Osnoven()`** во **`Izveden::~Izveden()`**
- Генерално, **деструкторот на изведената класа се повикува пред деструкторот на основната класа**

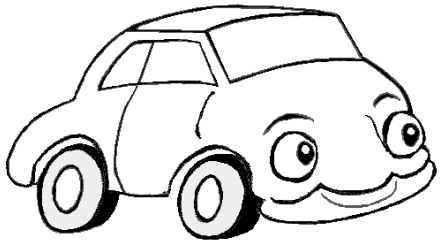
Деструктори на изведена класа (3)

```
#include <iostream>
using namespace std;
class BC {
public:
        BC() { sBC = new char[3]; cout << "BC allocates 3 bytes" << endl; }
        ~BC() { delete[] sBC;     cout << "BC frees 3 bytes" << endl; }
private: char* sBC;
};
class DC : public BC {
public:
        DC() : BC() { sDC = new char[5]; cout << "DC allocates 5 bytes" << endl; }
        ~DC() { delete[] sDC;     cout << "DC frees 5 bytes" << endl; }
private: char* sDC;
};
int main() {
        DC d;
        return 0;
}
```

```
BC allocates 3 bytes
DC allocates 5 bytes
DC frees 5 bytes
BC frees 3 bytes
```

Препокривање на функциите во изведената класа (1)

- **Во изведената класа може да се предефинираат функциите членки на основната класа**
  - изменетата дефиниција се однесува само на објектите од изведената класа
  - функцијата членка на изведената класа има исто име, како и функцијата членка на основната класа



Препокривање на функциите во изведената класа (2)

```
class Kamion : public Avtomobil {
public: // konstruktori
        Kamion();
        Kamion(int TezMasinskiDel, int sp, char const *nm, int PrikolkaTez);
        // interfejs za postavuvanje na dvete polinja za tezina
        void PostaviTezina(int engine_wt, int trailer_wt);
        // I vrakanje na vkupnata tezina
        int VratiTezina() const;
private: // podatoci
        int TezinaPrikolka;
};
// primer na konstruktor
Kamion::Kamion(int TezMasinskiDel, int sp, char const *nm, int trailer_wt) :
Avtomobil(TezMasinskiDel, sp, nm) {
        TezinaPrikolka = trailer_wt;
}
```

Препокривање на функциите во изведената класа (3)

- Класата **`Kamion`** **содржи две функции што се присутни во основната класа**
- **Функцијата `PostaviTezina()` е веќе дефинирана во `Avtomobil`**
  - препокривањето на истата во `Kamion` не претставува проблем
  - функционалноста едноставно е изменета за да се извршат акциите специфични за класата `Kamion`
- Дефиницијата на `PostaviTezina()` во класата `Kamion` **ќе ја покрие** верзијата на функцијата со исто име во `Avtomobil` (верзијата дефинирана во `Vozilo`)
- **За `Kamion` само функцијата `PostaviTezina()` со два аргумента** ќе биде употребена

Препокривање на функциите во изведената класа (4)

- **Функцијата `PostaviTezina()` на класата `Vozilo` може да се користи**
  - за да се повика мора да биде експлицитно повикана со користење на следниот израз `Avtomobil::PostaviTezina()`
  - Ова е потребно иако `Avtomobil::PostaviTezina()` има само еден целоброен аргумент

```
void Kamion::PostaviTezina(int engine_wt, int PrikolkaTez) {
  TezinaPrikolka = PrikolkaTez;
  Avtomobil::PostaviTezina(engine_wt);
  // zabeleska Avtomobil:: e potrebno
}
```

- **Истото се случува и за функцијата `VratiTezina()`** која е дефинирана во класата `Vozilo`, со иста листа на аргументи како и во класата `Kamion`

```
int Kamion::VratiTezina() const {
  return (Avtomobil::VratiTezina() + TezinaPrikolka);
}
```

Препокривање на функциите во изведената класа (5)

- Следниот пример ја илустрира примената на класите

```
int main() {
   Zemjeno veh(1200, 145);
   Kamion lorry(3000, 120, "Juggernaut", 2500);
   lorry.Vozilo::PostaviTezina(4000);
   cout << endl;
   cout << "Kamionot tezi " << lorry.Vozilo::VratiTezina() << endl;
   cout << "Kamionot + prikolkata tezat " << lorry.VratiTezina() << endl;
   cout << "Negovata brzina iznesuva " << lorry.VratiBrzina() << endl;
   cout << "Kamionot e od markata " << lorry.VratiIme() << endl;
   return (0);
}
```

приколка

- `Vozilo::PostaviTezina(4000)` се повикува експлицитно за да се изврши функцијата `Vozilo::PostaviTezina()`, што е дел од множеството функции на класата `Vozilo`, иако `Vozilo::PostaviTezina()`, може да се набљудува како преоптоварена верзија `Kamion::PostaviTezina()`

Препокривање на функции - пример

```
#include <iostream>
using namespace std;
class BC {
public:
        void g(int) { cout << "BC::g()" << endl; }
        void h(double) { cout << "BC::h()" << endl; }
};
class DC : public BC {
public:
        void g(int) { cout << "DC::g()" << endl; }
        void h(char[]) { cout << "DC::h()" << endl; }
};
int main() {
        DC d;
        d.h("Boffo!");
        d.h(707.7); //Error!
        d.BC::h(707.7);
        d.g(3);
        d.BC::g(4);
        return 0;
}
```

`Protected` членови на класата (1)

- **Ако изведената класа има потреба**
  - **да пристапи до приватните податочни членови на основната класа,** но
  - исто така е потребно **да се оневозможи пристап до овие членови на сите останати функции**
- **тогаш** членовите се декларираат како “`protected`”
- `Protected` **значи сите изведени класи** од основната **имаат пристап** до овие податоци или функции, **но тие во основа остануваат** `private`

Protected членови на класата (2)

```
#include <iostream>
using namespace std;
class BC {
 public:       void setX(int x) { this->x = x; }
 protected:    int getX() const { return x; }
 private:      int x;
};
class DC : public BC {
 public:       void add2() { int c = getX(); setX(c + 2); }
};
int main() {
        DC d;
        d.setX(3);
        cout << d.getX() << endl;  // Error!
        d.x = 77;  // Error!
        d.add2();
        return 0;
};
```

Права на пристап при изведувањето (1)

- Ако одредена класа е изведена од дадена базна класа со **`public` изведување**
  - `public` членовите од базната класа остануваат `public` членови во изведената класа
  - `protected` членовите од базната класа остануваат `protected` членови во изведената класа
- Ако одредена класа е изведена од дадена базна класа со **`protected` изведување**
  - `public` и `protected` членовите од базната класа стануваат `protected` членови во изведената класа
- Ако одредена класа е изведена од дадена базна класа со **`private` изведување**
  - `public` и `protected` членовите од базната класа стануваат `private` членови во изведената класа

Права на пристап при изведувањето (2)

во базна класа	изведување		
	public	**protected**	**private**
protected	protected	protected	private
public	public	protected	private