Curs 6

- **Polimorfism – Metode pur virtuale, Clase abstracte**
- **Operatii de intrări ieșiri în C++**
  - **Fișire**
- **Excepții**

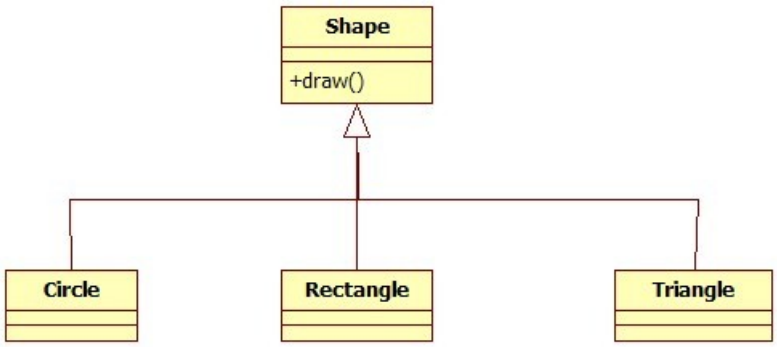
Funcții pur virtuale

Funcțiile pur virtuale nu sunt definite (avem doar declarația metodei).
Folosim metode pur virtuale pentru a ne asigura că toate clasele derivate (concrete) o sa definească metoda.

```
class Shape {
public:
     Shape();
     virtual ~Shape();
     virtual void draw() = 0;//pure virtual
};
```

=0 indică faptul ca nu există implementare pentru această metodă în clasă.
Clasele care au metode pur virtuale nu se pot instanția

Shape este o clasă abstractă - definește doar interfața, dar nu conține implementări.



Clase abstracte

O clasă abstractă poate fi folosită ca și clasă de bază pentru o colecție de clase derivate;
Oferă:

- o interfață comună pentru clasele derivate (metodele pur virtuale se vor implementa în clasele derivate)
- pot conține atribute comune tuturor claselor derivate

o clasă abstractă nu are instațe
o clasă abstractă are cel puțin o metodă pur virtuală:
`virtual` <return-type> <name> (<parameters>) = 0;

clasă pur abstractă = clasă care are doar metode pur vortuale
clasă pur abstractă = interfață

În UML font italic

Clase care extind clase abstracte

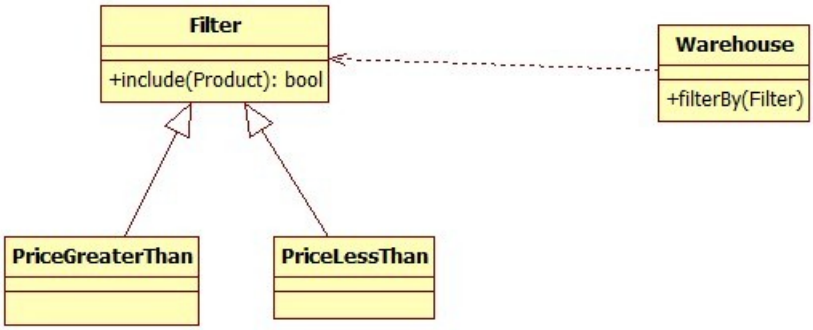
- O clasă derivată dintr-o clasă abstractă mosteneste interfața publică a clasei abstracte
- clasa suprascrie metodele definite în clasa abstractă, oferă implementări specifice pentru funcțiile definite în clasa abstracta
- putem avea instanțe

Moștenire. Polimorfism

Avantaje:

- **reutilizare de cod**
  - **clasa derivată moștenește din clasa de bază**
  - **se evită copy/paste – mai ușor de intreținut,înțeles**
- **extensibilitate**
  - **permite adaugarea cu ușurință de noi funcționalități**
  - **extindem aplicația fără să modificăm codul existent**

Exemplu : Filrare produse




```
DynamicArray3<Product*>* Warehouse::filterByPrice(double price) {
	DynamicArray3<Product*>* rez = new DynamicArray3<Product*>();

	DynamicArray3<Product*>* all = repo->getAllProds();

	for (int i = 0; i < all->getSize(); i++) {
		Product* p = all->get(i);
		if (p->getPrice() > price) {
			//make a copy of the product
			rez->addE(new Product(*p));
		}
	}
	return rez;
}
```

```
DynamicArray3<Product*>* Warehouse::filterByPriceLT(double price) {
	SmallerThanPrice* f = new SmallerThanPrice(price);
	return filterBy(f);
}

DynamicArray3<Product*>* Warehouse::filterByPriceGT(double price) {
	GreaterThanPrice* f = new GreaterThanPrice(price);
	return filterBy(f);
}

DynamicArray3<Product*>* Warehouse::filterBy(Filter* filter) {
	DynamicArray3<Product*>* rez = new DynamicArray3<Product*>();
	DynamicArray3<Product*>* all = repo->getAllProds();
	for (int i = 0; i < all->getSize(); i++) {
		Product* p = all->get(i);
		//polimorphic method invocation
		if (!filter->include(p)) {
			//make a copy of the product
			rez->addE(new Product(*p));
		}
	}
	return rez;
}
```

Operații de intrare/ieșire

IO (Input/Output) în C

<stdio.h> -> **scanf**(), **printf**(), **getchar**(), **getc**(), **putc**(), **open**(), **close**(), **fgetc**(), etc.

- Funțiile din C nu sunt extensibile
- funcționează doar cu un set limitat de tipuri de date (**char**, **int**, **float**, **double**).
- Nu fac parte din librăria standard => Implementările pot diferi (ANSI standard)
- pentru fiecare clasă nouă, ar trebui sa adăugăm o versiune nouă (supraâncărcare) de funcții **printf**() and **scanf**() și variantele pentru lucru cu fișiere, șiruri de caractere
- Metodele supraâncarcate au același nume dar o listă de parametrii diferit. Metodele printf și variantele pentru string, fișier folosesc o listă de argumente variabilă – nu putem supraâncărca.

Biblioteca de intrare/ieșire din C++ a fost creat să:

- fie ușor de exstins
- ușor de adăugat/folosit tipuri noi de date

I/O streams. I/O Hierarchies of classes.

Iostream este o bibliotecă folosit pentru operații de intrări ieșiri in C++.
Este orientat-obiect și oferă operații de intrări/ieșiri bazat pe noțiunea de flux (stream)

iostream este parte din C++ Standard Library și conține un set de clase template și funcții utile in C++ pentru operații IO

Biblioteca standard de intrări/ieșiri (iostream) conține:

Clase template

O hierarhie de clase template, implementate astfel încât se pot folosi cu orice tip de date.

Instanțe de clase template

Biblioteca oferă instațe ale claselor template speciale pentru manipulare de caractere **char** (narrow-oriented) respectiv pentru elemente de tip wchar (wide-oriented).

Obiecte standard

În fișierul header **<iostream>** sunt declarate obiecte care pot fi folosite pentru operații cu intrare/ieșire standard.

Tipuri

conține tipuri noi, folosite biblioteca standard, cum ar fi: streampos, streamoff and streamsize (reprezintă poziții, offset, dimensiuni)

Manipulatori

Funții globale care modifică proprietăți generale, oferă informații de formatare pentru streamuri.

Flux - Stream

Noțiunea de flux este o abstractizare, reprezintâ orice dispozitiv pe care executăm operații de intrări / ieșiri (citire/scriere)

Stream este un flux de date de la un set de surse (tastatură, fișier, zonă de memorie) către un set de destinații (ecran, fișier, zonă de memorie)

În general fiecare stream este asociat cu o sursă sau destinație fizică care permite citire/scriere de caractere.

De exemplu: un fișier pe disk, tastatura, consola. Caracterele citite/scrise folosind streamuri ajung / sunt preluate de la dispozitive fizice existente (hardware).
Stream de fisiere - sunt obiecte care interacționează cu fișiere, dacă am atașat un stream la un fișier orice operație de scriere se reflectă în fișierul de pe disc.

Buffer

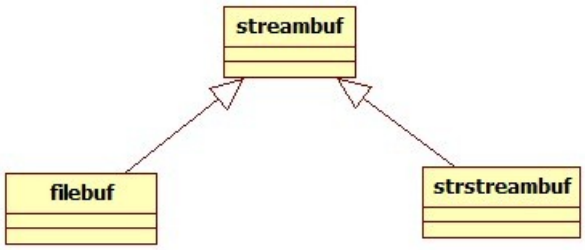
buffer este o zonă de memore care este un intermediar între stream și dispozitiv.

De fiecare dată când se apelează metoda put (scrie un caracter), caracterul nu este trimis la dispozitivul destinație (ex. Fișier) cu care este asociat streamul. Defapt caracterul este inserat în buffer

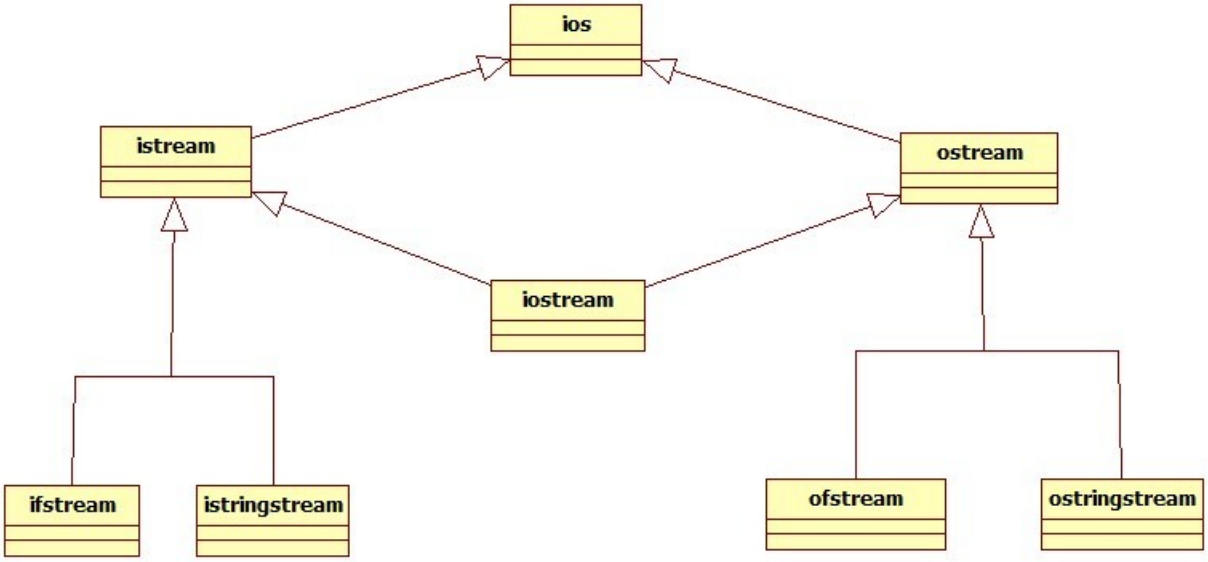
Când bufferul este golit (flush) , toate datele se trimit la dispozitiv (daca era un strim de ieșire). Procesul prin care conținutul bufferului este trimis la dispozitiv se numeste sincronizare și se întâmplă dacă:

- streamul este închis, toate datele care sunt în buffer se trimit la dispozitiv (se scriu ân fișier, se trimite la consolă, etc.)
- bufferul este plin. Fiecare buffer are o dimensiune, dacă se umple atunci se trimite conținutul lui la dispozitiv
- programatorul poate declanșa sincronizarea folosind manipulatoare: **flush**, **endl**
- programatorul poate declanșa sincronizarea folosind metoda **sync()**

Fiecare obiect stream din biblioteca standard are atașat un buffer (**streambuf**)



Hierarhie de clase din biblioteca standard IO C++



Fișiere header din IOStream

Clasele folosite pentru intrări/ieșiri sunt definite în fișiere header:

- <ios> formatare , streambuffer.
- <istream> intrări formatate
- <ostream> ieșiri formatate
- <**iostream**> implementează intrări/ieșiri formatate
- <fstream> intrări/ieșiri fișiere.
- <sstream> intrări/ieșiri pentru streamuri de tip string.
- <iomanip> conține manipulatori.
- <iosfwd> declarații pentru toate clasele din biblioteca IO.

Streamuri standard – definite în <iostream>

cin - corespunde intrării standard (stdin), este de tip **istream**

cout - corespunde ieșirii standard (stdout) , este de tip **ostream**

cerr - corespunde ieșirii standard de erori (stderr), este de tip **ostream**

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

void testStandardIOStreams() {
	cout << "!!!Hello World!!!" << endl; // prints !!!Hello World!!! to
the console
	int i = 0;
	cin >> i; //read an int from the console
	cout << "i=" << i << endl; // prints !!!Hello World!!! to the console

    cerr << "Error message";//write a message to the standard error stream
}
```

Operatorul de inserție (Insertion operator - output)

- Pentru operațiile de scriere pe un stream (ieșire standard, fișier, etc) de folosește operatorul "<<", numit operator de inserție

- pe partea stângă trebuie sa avem un obiect de tip ostream (sau derivat din ostream). Pentru a scrie pe ieșire standard (consolă) se folosește **cout** (declarat in modulul iostream)

- pe dreapta putem avea o expresie.

- Operatorul este supraîncărcat pentru tipurile standard, pentru tipurile noi programatorul trebuie sa supraîncarce.

```
void testWriteToStandardStream() {
	cout << 1 << endl;
	cout << 1.4 << endl << 12 << endl;
	cout << "asdasd" << endl;
	string a("aaaaaaaa");
	cout << a << endl;

	int ints[10] = { 0 };
	cout << ints << endl; //print the memory address
}
```

- Se pot înlănțui operații de inserție, evaluarea se face în ordinea inversă scrierii. Înlănțuirea funcționează fiindca operatorul << returnează o referință la stream

Operatorul de extragere – citire (Extraction operator)

- Citirea dintr-un stream se realizează folosind operatorul “>>“
- operandul din stânga trebuie să fie un obiect de tip istream (sau derivat din istream). Pentru a citi din intrarea standard (consolă) putem folosi cin, obiect declarat în iostream
- operandul de pe dreapta poate fi o expresie, pentru tipuri standard operatorul de extragere este supraâncărcat.
- programatorul poate supraâncărca operatorul pentru tipuri noi.

```
void testStandardInput() {
	int i = 0;
	cout << "Enter int:";
	cin >> i;
	cout << i << endl;
	double d = 0;
	cout << "Enter double:";
	cin >> d;
	cout << d << endl;
	string s;
	cin >> s;
	cout << s << endl;
}
```

Supraîncărcare operatori <<, >> pentru tipuri utilizator

- Se face similar ca și pentru orice operator
- sunt operatori binari
- primul operand este un stream (pe stânga), pe partea dreaptă avem un obiect de tipul nou (user defined).

```
class Product {
public:
	Product(int code, string desc,
double price)
	~Product();
	double getCode() const {
		return code;
	}
	double getPrice() const {
		return price;
	}
	const string& getDescription()
const {
		return description;
	}

	friend ostream& operator<<
(ostream& stream, const Product&
prod);

private:
	int code;
	string description;
	double price;
};
```

```
ostream& operator<<(ostream&
stream, const Product& prod) {
  stream << prod.code << " ";
  stream  << prod.description << "
";
  stream  << prod.price;
  return stream;
}
```

```
void testStandardOutputUserType() {
	Product p = Product(1, "prod", 21.1);
	cout << p << "\n";
	Product p2 = Product(2, "prod2", 2.4);
	cout << p2 << "\n";
}
```

Formatare scriere

width(int x)	Numarul minim de caractere pentru scrierea următoare
fill(char x)	Caracter folosit pentru a umple spațiu dacă e nevoie să completeze cu caractere (lungime mai mică decât cel setat folond width).
precision(int x)	Numărul de zecimale scrise

```
void testFormatOutput() {
	cout.width(5);
	cout << "a";
	cout.width(5);
	cout << "bb" << endl;
	const double PI = 3.1415926535897;
	cout.precision(3);
	cout << PI << endl;
	cout.precision(8);
	cout << PI << endl;
}
```

Manipulatori.

- Manipulatorii sunt funcții cu semantică specială, folosite înpreună cu operatorul de inserare/extragere (<< , >>)

- Sunt funcții obișnuite, se pot și apela (se da un argument de tip steam)

- Manipulatorii se folosesc pentru a schimba modul de formatare a streamului sau pentru a insera caractere speciale.

- Există o variabilă membră în ios (x_flags) care conține informații de formatare pentru operare I/O , x_flags poate fi modificat folosind manipulatori

- sunt definite în modulul **iostream.h** (endl, dec, hex, oct, etc) și **iomanip.h** (setbase(int b),setw(int w),setprecision(int p))

```
void testManipulators() {
     cout << oct << 9 << endl << dec << 9 << endl;
     oct(cout);
     cout << 9;
     dec(cout);
}
```

Flag-uri

Indică starea internă a unui stream:

Flag	Descriere	Metodă
fail	Date invalide	fail()
badbit	Eroare fizică	bad()
goodbit	OK	good()
eofbit	Sfârșid de stream detectat	eof()

```
void testFlags(){
	cin.setstate(ios::badbit);
	if (cin.bad()){
		//something wrong
	}
}
```

Flag de control :

```
cin.setf(ios::skipws);  //Skip white space. (For input; this is the default.)
cin.unsetf(ios::skipws);
```

Fișiere

Pentru a folosi fișiere on aplicații C++ trebuie sa conectăm streamul la un fișier de pe disk

fstream oferă metode pentru citere/scriere date din/in fișiere.

<fstream.h>

- ifstream (input file stream)
- ofstream (output file stream)

Putem atașa fișierul de stream folosind constructorul sau metoda **open**

După ce am terminat operațiile de IO pe fișier trebuie sa închidem (deasocien) streamul de fișier folosind metoda close. Ulterior, folosind metoda **open,** putem folosi streamul pentru a lucra cu un alt fișier.

Metoda **is_open** se poate folosi pentru a verifica daca streamul este asociat cu un fișier.

Output File Stream

```
#include <fstream>

void testOutputToFile() {
  ofstream out("test.out");
  out << "asdasdasd" << endl;
  out << "kkkkkkk" << endl;
  out << 7 << endl;
  out.close();
}
```

- Dacă fișierul “test.out” există pe disc, se deschide fișierul pentru scriere și se conecteaza streamul la fișier. Conținutul fișierului este șters la deschidere.

- Daca nu există “test.out”: se crează, se deschide fișierul pentru scriere și se conecteaza streamul la fișier.

Input File Stream

```
void testInputFromFile() {
	ifstream in("test.out");
	 //verify if the stream is
opened
	 if (in.fail()) {
		return;
	}
	while (!in.eof()) {
		string s;
		in >> s;
		cout << s << endl;
	}
	in.close();
}
```

```
void testInputFromFileByLine() {
	ifstream in;
	in.open("test.out");
	 //verify if the stream is
opened
	if (!in.is_open()) {
		return;
	}
	while (in.good()) {
		string s;
		getline(in, s);
		cout << s << endl;
	}
	in.close();
}
```

- Dacă fișierul “test.out” există pe disc, se deschide pentru citire și se conectează streamul la fișier.
- Dacă nu există fișierul streamul nu se asociează, nu se poate citi din stream
- Unele implementari de C++ crează fișier dacă acesta nu există.

Open

Funcția open deschide fișierul și asociază cu stream-ul:

open (filename, mode);

filename sir de caractere ce indică fișierul care se deschide
mode parametru optional, indică modul în care se deschide fișierul. Poate fo o combinație dintre următoarele flaguri:

ios::in	Deschide pentru citire.
ios::out	Deschide pentru scriere.
ios::binary	Mod binar.
ios::ate	Se poziționează la sfârșitul fișierului. Dacă nu e setat,după deschidere se poziționează la început.
ios::app	Toate operațiile de scriere se efectuează la sfărșitul fișierului, se adaugă la conținutul existent. Poate fi folosit doar pe stream-uri deschise pentru scriere.
ios::trunc	Sterge conținutul existent.

Flag-urile se pot combine folosind operatorul pe biți OR (|).

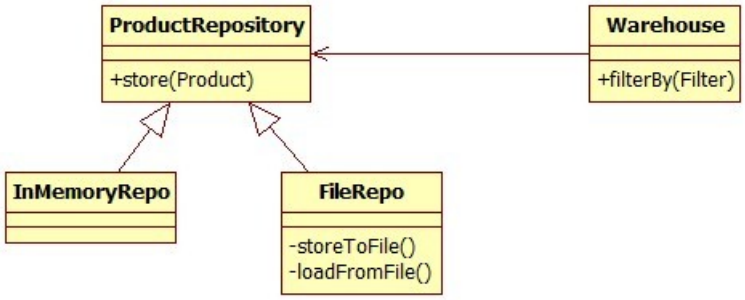
Citire/scriere obiecte

```
void testWriteReadUserObjFile() {

	ofstream out;
	out.open("test2.out", ios::out | ios::trunc);
	if (!out.is_open()) {
		return;
	}
	Product p1(1, "p1", 1.0);
	out << p1 << endl;
	Product p2(2, "p2", 2.0);
	out << p2;
	out.close();

	//read
	ifstream in("test2.out");
	if (!in.is_open()) {
		cout << "Unable to open";
		return;
	}
	Product p(0, "", 0);
	while (!in.eof()) {
		in >> p;
		cout << p << endl;
	}
	in.close();
}
```

Exemplu: FileRepository



Varianta 2