Тема 14

	[illegible]
14	Type casting. SOLID principles. Design patterns – типове и примери (singleton, factory, prototype, composite, flyweight, iterator, command, visitor). [illegible] singleton + factory
15	[illegible]

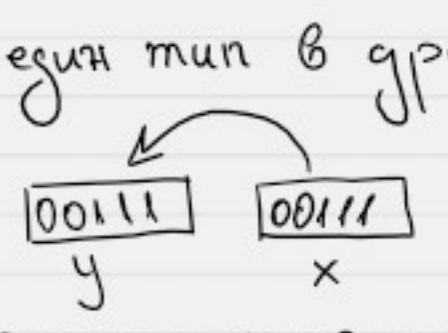
Type casting

def – експлицитно конвертиране на израз от един тип в друг

Видове:

1) преобразуване от един тип в друг

```
int x=7;
double y=x;
```



- данните от x се копират и се интерпретират като y

2) преобразуване на указател
(вместо да попълваме данни, както в 1))

Base
↑
Der

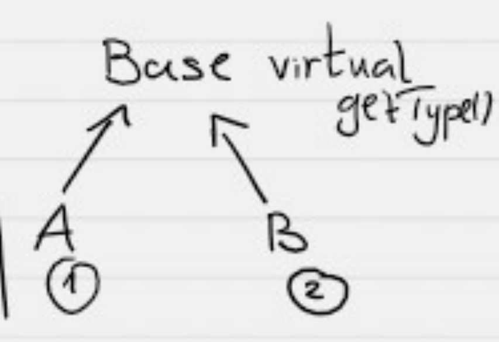
```
Der* d =new D();
Base* b=d; // Base* b =(Base*)d
```

За потребителски дефинизани типове са въведени 4 типа type-cast. Те са шаблонни класове приемащи обект и връщащи обект от новия тип

Общ синтаксис: type-of-casting <type_to_cast to>(object_to_cast);

1. static_cast - използва се за преуобразуване на ук-л/реф. към базов клас към ук-л/реф. към клас наследник

```
f(Base* ptr) {
    if(ptr -> getType() == 1)
        A* obj =static_cast <A*>(ptr);   // downcasting
```



! Не прави run-time check, затова го използваме само когато сме сигурни, че ще преобразуваме в правилен за указателя тип

→ Ако се опитаме да кастнем към грешен тип, се очаква да крашне. Ако не даде грешка, е undefined behaviour

```
Base* ptr= new A();
ptr =static_cast <B*>(ptr); //грешка
```

void* - указател, който може да сочи към всичко, но не знаем към какво сочи
- не можем да го дереферираме или местим
- за да направим едно от двете трябва да го кастнем

2. dynamic_cast - преобразуване на ук-л/реф. от базов клас към ук-л/реф от клас наследник, но с runtime check

- Основно се използва за downcasting, когато не знаем какъв е типа на обекта
- За upcasting не се използва, защото той винаги е успешен и няма нужда от проверката.

Run_time check на dynamic_cast

Downcasting-а може да не е възможен, ако:

- се опитваме да преобразуваме към грешен тип;
- Base указателят всъщност не сочи към наследник.

Тази проверка става през виртуалната таблица, затова downcasting можем да правим само ако имаме поне една виртуална функция в базовия клас

При успешно преобразуване:

- на ук-л връща указател от желания тип;
- на референция връща нова референция от желания тип.

При неуспешно преобразуване:

- на указател dynamic_cast връща nullptr;
- на референция dynamic_cast хвърля std::bad_cast.
- за обекти, които не са в полиморфна йерархия връща nullptr/bad_cast

3. const_cast - манипулира константността на обекти, с които работим през ук-ли или референции

За да премахнем константността трябва обектът да не е бил константен при създаването си.
В противен случай при опит за const_cast имаме undefined behaviour.

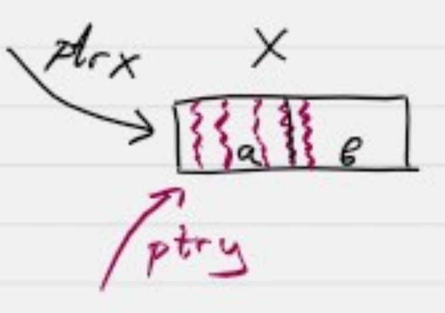
Използването му не е добра практика.

4. reinterpret_cast - преобразуване на указател от произволен тип към указател от произволен тип

Реинтерпретираме паметта по нов начин

```
class X{            class Y{
  int a;              char ch[5];
  int b;
};                  }

X* ptrx = new X();
Y* ptry = reinterpred_cast <Y*>(ptrx);
```



C-style cast

```
A* ptr = (B*)ptr2;
```

- const_cast
- static_cast
- static_cast + const_cast
- reinterpred_cast
- reinterpred_cast + const_cast

↓ Проверява дали типовете са подходящи за видовете кастване в този ред