# Язык C++

## ООП. Наследование

# Наследование

```
class CPerson {
public:
  CPerson(const std::string& name, unsigned yearOfBirth)
      : yearOfBirth_(yearOfBirth)
      , name_(name)   {}

  unsigned age() const {
       const std::chrono::time_point now{std::chrono::system_clock::now()};
       const std::chrono::year_month_day ymd{std::chrono::floor<std::chrono::days>(now)};

       return static_cast<int>(ymd.year()) - yearOfBirth_;
  }
  const std::string& name() const {
      return name_;
  }
 private:
  std::string name_;
  unsigned yearOfBirth_;
};
```

# Наследование

- позволяет описать новый класс на основе уже существующего с частично или полностью заимствующейся функциональностью. Класс, от которого производится наследование, называется базовым, родительским или суперклассом. Новый класс — потомком, наследником, дочерним или производным классом.
- полиморфизм подтипов, is-a relationship
- обеспечивает повторное использование кода (следствие но не причина)
- множественное наследование

# Наследование

```
class CStudent : public CPerson {
public:
   CStudent(const std::string& name, unsigned age, const std::string& university)
       : CPerson(name, age)
       , university_(university)
   {}
private:
   std::string university_;
};
```

# Наследование

Наследник:

- Хранит в себе родителя
- Сохраняет методы родителя*
- Приведение к базовому классу (slicing)
- Модификаторы доступа

# Constructor\Destructor order

# Наследование

| Specifiers | Within Same Class | In Derived Class | Outside the Class |
|---|---|---|---|
| Private | Yes | No | No |
| Protected | Yes | Yes | No |
| Public | Yes | Yes | Yes |

Sitesbay.com

# Наследование

```
class CStudent : public CPerson {
public:
    CStudent(const std::string& name, unsigned yearOfBirth, const std::string& university)
        : CPerson(name, yearOfBirth)
        , university_(university)
    {}

    const std::string& university() const {
        return university_;
    }

    void Hello() const {
        std::cout << "Hello. I'am " << name() << " I'am from " << university_ << std::endl;
    }

private:
    std::string university_;
};
```

# Наследование

```
// CBudgetStudent is a CStudent. CStudent is a CPerson

class CBudgetStudent : public CStudent {
public:
    CBudgetStudent(const std::string& name, unsigned yearOfBirth,
                   const std::string& university, unsigned sallary)
       : CStudent(name, yearOfBirth, university)
       , sallary_(sallary)
    {}

private:
   unsigned sallary_;
};
```

# is-a relationship

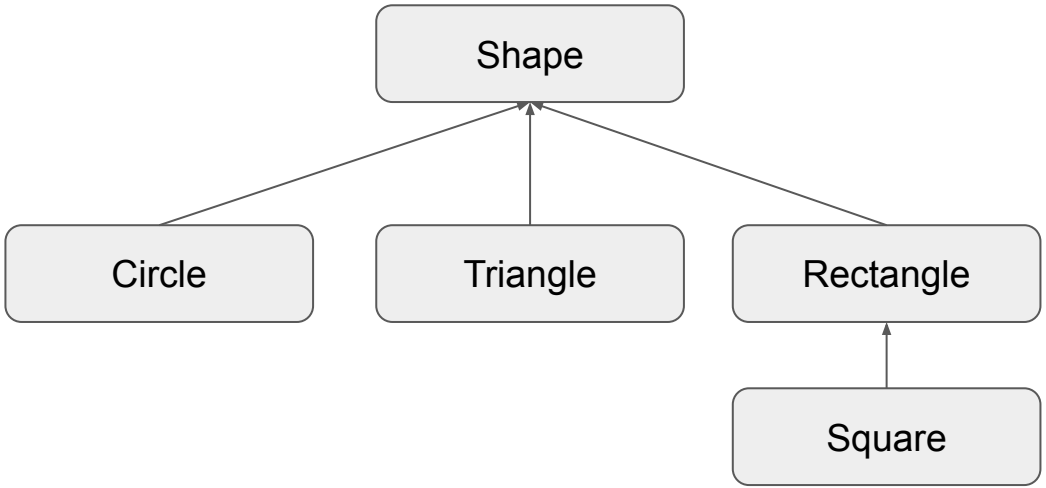
```
void Hello(const CStudent& p) {
    p.Hello();
}

int main() {
    CBudgetStudent st = {"Ivan Ivanov", 2002, "ITMO", 20000};
    Hello(st);
    return 0;
}
```

# Наследование, устройство в памяти

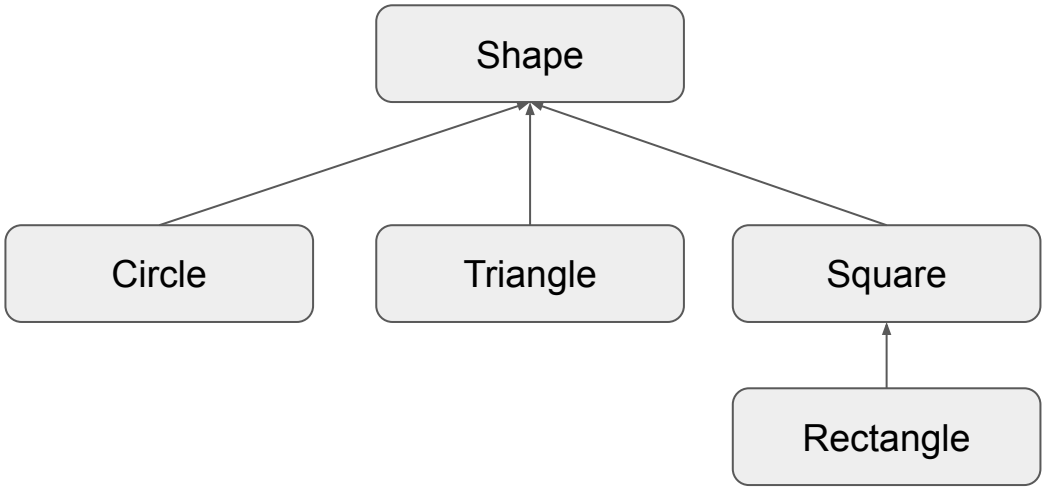
```
int main(int, char**) {
   std::cout << "sizeof(CPerson): " << sizeof(CPerson) << std::endl;
   std::cout << "sizeof(CStudent): " << sizeof(CStudent) << std::endl;
   std::cout << "sizeof(CBudgetStudent): " << sizeof(CBudgetStudent) <<
std::endl;
}
```

# Иерархия геометрических фигур



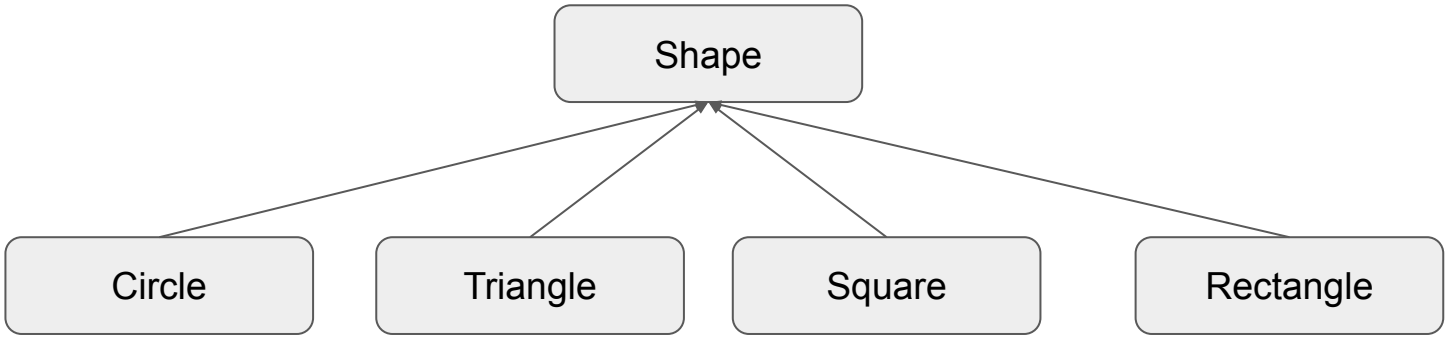

```
void double_width(Rectangle& r)
{
  r.setWidth(r.width*2);
}
```

# Иерархия геометрических фигур




```
double area(Square& s) {
  return s.width() * s.width();
}
```

# Иерархия геометрических фигур



# Множественное наследование

```
class CEmployee : public CPerson {
public:
   CEmployee(const std::string& name, int yearOfBirth, unsigned sallary)
       : CPerson(name, yearOfBirth)
       , sallary_(sallary)
   {}

private:
   unsigned sallary_ ;
};
```

# Множественное наследование

```
class CIntern : public CEmployee, public CBudgetStudent {
public:
   CIntern(
       const std::string& name,
       int yearOfBirth,
       const std::string& university,
       unsigned universSallary,
       unsigned workSallary
   )
       : CEmployee(name, yearOfBirth, workSallary)
       , CBudgetStudent(name, yearOfBirth, university, universeSallary)
   {}
};
```

# Множественное наследование

```
int main(int, char**) {
    std::cout << "sizeof(CPerson): " << sizeof(CPerson) << std::endl;
    std::cout << "sizeof(CStudent): " << sizeof(CStudent) << std::endl;
    std::cout << "sizeof(CBudgetStudent): " << sizeof(CBudgetStudent) << std::endl;

    std::cout << "sizeof(CEmployee): " << sizeof(CEmployee) << std::endl;
    std::cout << "sizeof(CIntern): " << sizeof(CIntern) << std::endl;
}
```

# Diamond Problem

```
int main(int, char**) {
    CIntern intern("Ivan Ivanov", 2002, "ITMO", 20000, 50000);

    intern.Hello();

    // std::cout << intern.name();  compile-time error

    std::cout << intern.CEmployee::name() << std::endl;
    std::cout << intern.CBudgetStudent::name() << std::endl;
    return 0;
}
```

# Проблемы множественного наследования

```
class CEmployee : public CPerson {
public:
   void IncreaseSallary() {
       sallary_ += 1000;
   }
protected:
   unsigned sallary_ ;
};

class CBudgetStudent : public CStudent {
public:
   void IncreaseSallary() {
       sallary_ += 1000;
   }

protected:
  unsigned sallary_;
};
```

```
class CIntern : public CEmployee, public CBudgetStudent {
public:
   using CEmployee::IncreaseSallary;

   unsigned Sallary() const {
       return CEmployee::sallary_ + CBudgetStudent::sallary_;
   }
};

int main(int, char**) {
   CIntern intern("Ivan Ivanov", 2002, "ITMO", 20000, 50000);

   intern.IncreaseSallary();
   std::cout << intern.Sallary() << std::endl;

   return 0;
}
```

# final