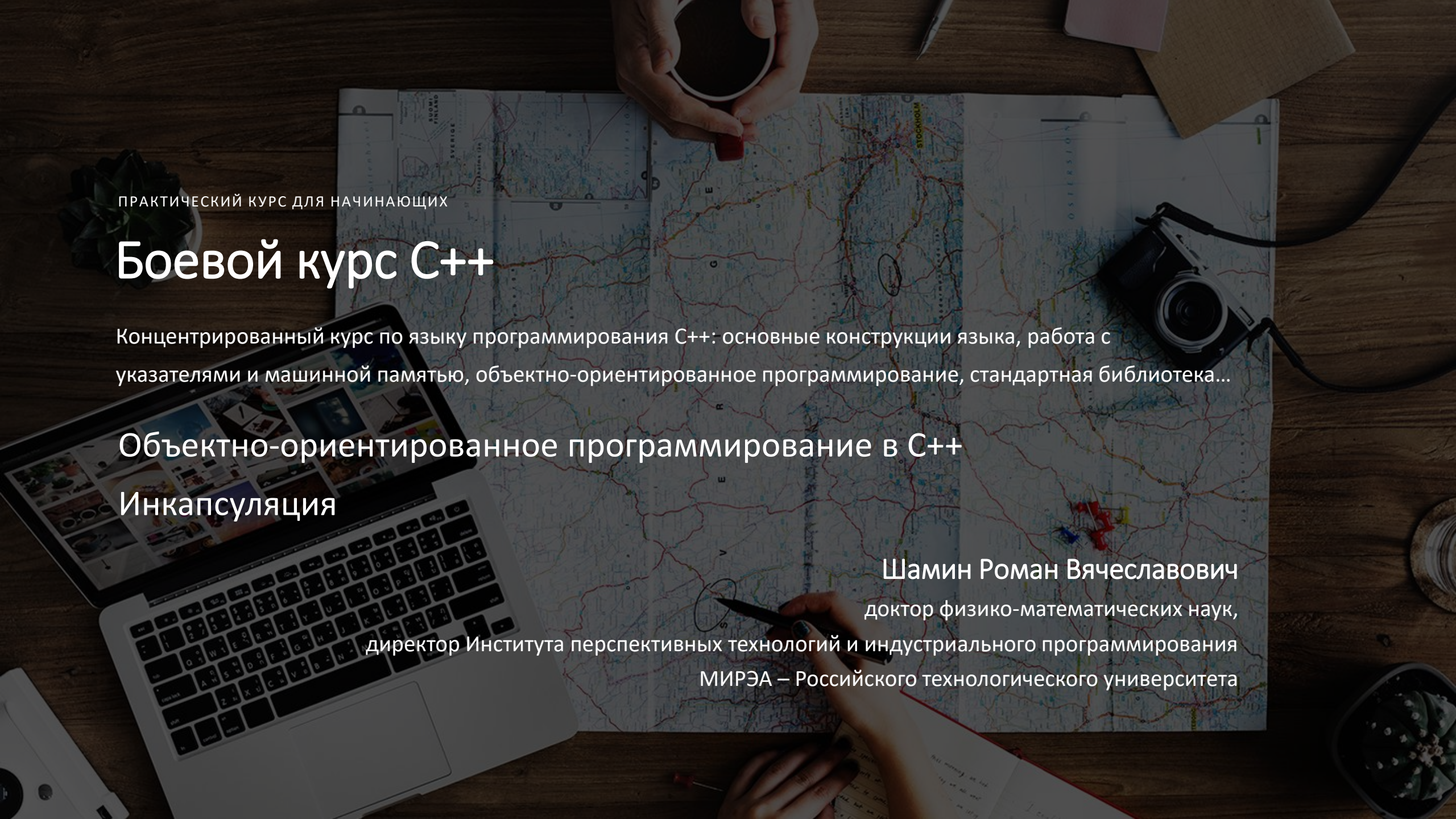


ПРАКТИЧЕСКИЙ КУРС ДЛЯ НАЧИНАЮЩИХ

Боевой курс C++

Концентрированный курс по языку программирования C++: основные конструкции языка, работа с указателями и машинной памятью, объектно-ориентированное программирование, стандартная библиотека…

Объектно-ориентированное программирование в C++

Инкапсуляция

Шамин Роман Вячеславович
доктор физико-математических наук,
директор Института перспективных технологий и индустриального программирования
МИРЭА – Российского технологического университета

Что такое классы?

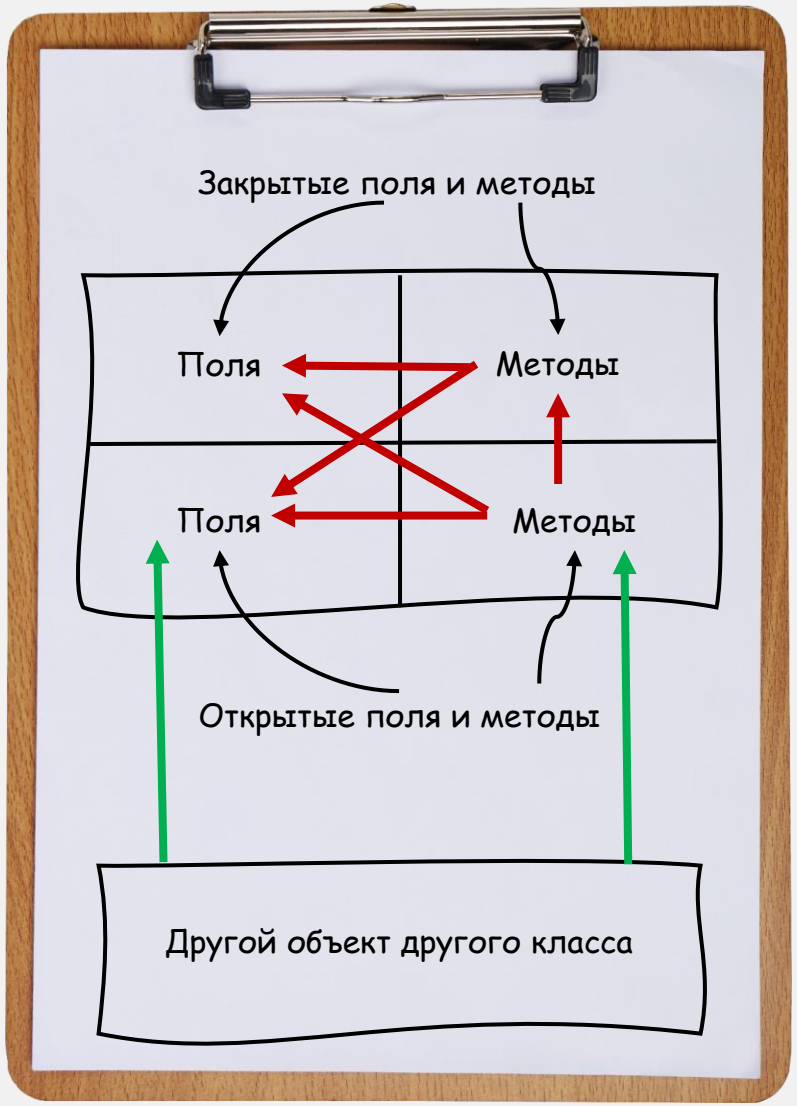
Объектно-ориентированное программирование основано на том, что все отдельные сущности (объекты) оформляются в виде классов. Каждый класс – это пользовательский тип данных, который хранит в себе как данные (набор различных переменных), так и функции для работы с этими данными. Данные в классе называются полями, а функции в классе называются методами. После определения класса (это будет новый тип данных) мы можем создавать объекты – переменные типа класса.

Важнейшим понятием объектно-ориентированного программирования является понятие инкапсуляции. **Инкапсуляция** – это объединение к одном объекте полей и методов, а также регламентация доступа к полям и методам из вне. Дело в том, что для обеспечения надежной логики работы с классами пользователь не ко всем полям и методам должен иметь доступ.

Принцип инкапсуляции позволяет работать с объектами как с черными ящиками, у которых есть интерфейс для «входа» и «выхода».

Другим принципиальным понятием объектно-ориентированного программирования является **наследование** – создание новых классов на базе уже существующих, которые будут наследовать все поля и методы предка.

Наконец, третьим китом объектно-ориентированного программирования является **полиморфизм**, с помощью которого при наследовании можно заменять методы на новые.



Посмотреть видео

Создание класса

Создадим класс для работы с персоной.

```
class TPers // Объявляем класс
{
public:
    string Name; // открытые поля
    int Year;
private:
    int Salary; // закрытое поле
public:
    TPers(string aName, int aYear, int aSalary) // конструктор обратите внимание на public
    {
        Name = aName; // сохраняем данные
        Year = aYear;
        Salary = aSalary;
    }

    void Print() // метод для печати
    {
        cout << "Name: " << Name << endl; // печатаем в нужном формате
        cout << "Year: " << Year << endl;
        cout << "Salary: " << Salary << endl;
    }
};
```

Мы создали новый тип данных – класс TPers и можем теперь определять переменные типа TPers.

Покажем как работать с созданным классом. Для этого объявим переменную типа класса и инициализируем ее, вызвав конструктор. При вызове конструктора мы передаем ему параметры. Заметим, что создать переменную типа класса, не вызывая конструктора нельзя, если у него есть параметры.

```
int main()
{
    setlocale(LC_ALL, "Russian"); // устанавливаем кириллицу

    TPers Pers = TPers("Роман", 1975, 120); // объявляем переменную и класса и
                                            // вызываем конструктор
    Pers.Print(); // вызываем метод
}
```

Для обращения к полям и методам класса используется оператор «.», но обычно создается не объекты типа класса, а указатели на объекты. В этом случае используется оператор «->» для доступа к полям и методам.

```
int main()
{
    TPers *P; // создаем указатель на класс
    P = new TPers("Сергей", 1991, 75); // выделяем память и вызываем конструктор
    P->Print(); // обращаемся к методу объекта
    delete P; // освобождаем память
}
```

Мы почти всегда будем использовать указатели на объекты и передавать только указатель на объект.

Передавая параметры конструктору, мы были вынуждены к имени переменной добавлять букву «а», что она не совпадала с именем поля. Но методы класса всегда имеют указатель на собственный объект – this. Вот как можно заменить наш конструктор.

```
TPers(string Name, int Year, int Salary) // конструктор имена параметров совпадают с полями
{
    this->Name = Name; // сохраняем данные, используя this
    this->Year = Year;
    this->Salary = Salary;
}
```

Покажем, как используя ключевое слово this, можно сделать копию своего объекта.

```
TPers* Copy() // создаем метод, который возвращает указатель на объект TPers
{
    TPers *P = new TPers(this->Name, this->Year, this->Salary); // создаем новый объект
    return P; // возвращаем указатель на новый объект
}
```

В дальнейшем мы еще увидим применение this, когда будем рассматривать перегрузку операторов.

Каждый экземпляр класса (созданный объект) имеет собственную копию своих полей. Однако в C++ есть конструкция, которая позволяет определить одно поле на все экземпляры класса. Это делается с помощью статических полей и методов.

Рассмотрим пример класса, который считает количество своих экземпляров.

```
class TLamp // объявляем класс
{
public:
    int Number; // номер каждого экземпляра
    static int Count; // статическое поле – общее количество экземпляров класса
    TLamp() // конструктор без параметров
    {
        Count++; // увеличиваем количество экземпляров
        Number = Count; // текущий номер присваиваем последнему объекту
    }
};
int TLamp::Count = 0; // статическое поле необходимо дополнительно объявить
```

Создаем три экземпляра и выводим для каждого номер и общее количество объектов.

```
TLamp* L1 = new TLamp(); // создаем новый экземпляр
TLamp* L2 = new TLamp(); // создаем новый экземпляр
TLamp* L3 = new TLamp(); // создаем новый экземпляр
cout << L1->Number << "\t" << L2->Number << "\t" << L3->Number << endl;
cout << "Всего: " << TLamp::Count << endl; // используем статическое поле
```

Скачать текст программы

Посмотреть видео