


Нижегородский государственный университет им. Н.И. Лобачевского

Институт информационных технологий, математики и механики

Нижегородский государственный университет им. Н.И. Лобачевского

Институт информационных технологий, математики и механики

Введение в Java

Классы Java в сравнении с C++.

Козинов Е.А.
Кафедра МОСТ

Содержание

- Классы и объекты.
- Спецификаторы доступа.
- Пакеты.
- Методы. Способы передачи данных.
- Конструкторы.
- Перегрузка операций.
- Jar архивы (в сравнении со статическими библиотеками C++).
  - Программы с несколькими точками входа.

КЛАССЫ И ОБЪЕКТЫ

Классы и объекты

- **Класс** – является абстрактным типом данных, определяемым пользователем, и представляет собой модель реального объекта в виде данных и операций над этими данными.
  - Класс – это «чертеж стола»
- **Реальный объект** – не стоит понимать буквально.
  - Стол – реальный объект
  - Контроллер базы данных – программная сущность
- **Важные вопросы**
  - Сколько может быть классов?
  - Сколько может быть объектов?
  - В каком отношении находятся классы и объекты?

Пример класса

- Пример класса точка на C++

```
// point.h

class Point
{
  int x, y;
public:
  Point(int x, int y);
  void move(int x, int y);
};
```

Объявление класса

```
// point.cpp
#include "point.h"
Point::Point(int x, int y)
{
  move(x, y);
}
void Point::move(int x, int y)
{
  this->x = x;
  this->y = y;
}
```

Реализация класса

Пример класса

- Пример класса на Java

```
// Point.java
class Point
{
  int x, y;
  Point(int x, int y)
  {
    move(x, y);
  }
  void move(int x, int y)
  {
    this.x = x;
    this.y = y;
  }
}
```

Имя класса
как правило один класс
один файл (имя совпадает)

Реализация класса
реализация не отделяется
от объявления класса

Обращение к полям
и методам через «.»

В конце «;» не ставятся

СПЕЦИФИКАТОРЫ ДОСТУПА

Спецификаторы доступа

- В Java существует четыре вида спецификатора доступа
  - public, private, protected, уровень доступа по умолчанию
- ***Где указываются спецификаторы доступа?***
  - У полей классов
  - У методов и конструкторов
  - ***У классов*** **(в C++ у классов уровня доступа нет)**
- Значение спецификаторов доступа
  - **public** – уровень доступа видимый для всех
  - **private** – уровень доступа видимый только для реализации класса
  - **protected** – уровень доступа видимый для пакета и всех подклассов
  - По умолчанию – уровень доступа без указания спецификатора
    В ***рамках пакета*** поля и методы и классы являются **public**
    Вне пакета поля и методы являются **private**

Инкапсуляция

- **class** – позволяет объединить данные и операции над ними в рамках одной синтаксической структуры языка программирования
- секции **public**, **private** и **protected** позволяют реализовать разграничения доступа, позволяя
  - скрыть детали внутренней реализации
  - организовать защиту данных
- ***Когда нужно скрывать поля?***
- ***Когда нужно скрывать методы?***

Скрытие данных

- Часто встречающаяся рекомендация
  - ***скрывайте все Ваши данные, объявляя их в секции private!***
- Что из этого следует? Если Вы хотите придерживаться этого тезиса, то Вам придется:
  - объявить все поля в секции **private**

```
private int x;
```

  - создать методы для доступа к этим полям

```
public void SetX(int x) {this.x = x;}
public int GetX() {return x;}
```

  - в противном случае, Вы не сможете работать с полями
- ***Всегда ли скрытие данных оправдано?***

```
public int GetX() {return if(x>0) return x;
                          else return -x;}
```

Скрытие методов

- К скрытию методов нужно подходить с умом. Рекомендации таковы:
  - Скрывайте те методы, которые имеют отношение к деталям внутренней реализации
  - Скрывайте методы, которые с Вашей точки зрения не понадобятся пользователю разработанного Вами класса
  - Методы **public** – интерфейс Вашего класса. Чем меньше там написано, тем проще пользователю класса разобраться, как с ним работать
    - В Java есть отдельное понятие интерфейс (будет рассмотрено в следующей лекции)
  - Создавайте те методы, которые нужны
    - Не создавайте много методов только потому, что это красиво смотрится. Пользователь класса обязательно запутается
    - Если необходимо много методов, то реализуйте java-интерфейс
    - Если необходимо много методов, то подумайте о декомпозиции задачи или смене интерфейса

Пример класса (со спецификаторами доступа)

- Пример класса на Java

Класс открытый
Открытый класс
в файле может быть только
один, его имя должно
совпадать с именем файла

Поля сделаны скрытыми

Реализация методов и конструкторов как правило открытая

```
// Point.java
public class Point
{
   private int x, y;
   public Point(int x, int y)
   {
      move(x, y);
   }
   public void move(int x, int y)
   {
      this.x = x;
      this.y = y;
   }
}
```

ПАКЕТЫ В JAVA

Понятие пакетов в Java

- Аналог пакетов в C++ пространства имен
  - Для каких целей используют пространства имен?
  - Может ли быть несколько пространств имен в одном файле?
  - Могут ли пространства имен быть вложенными?

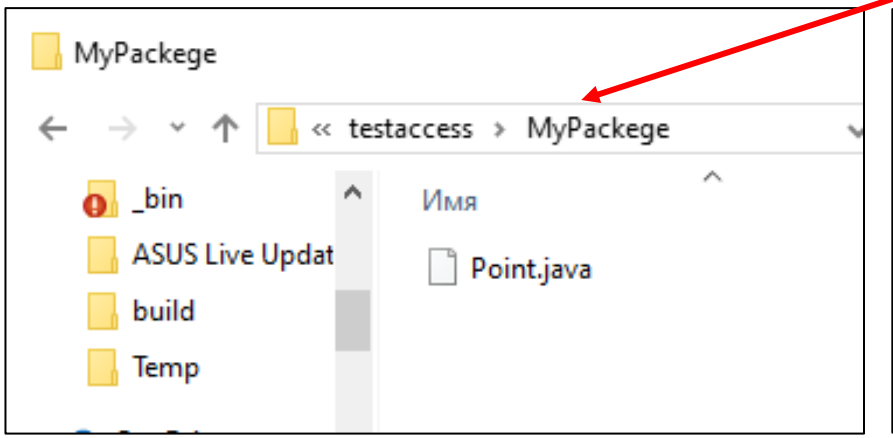
```
namespace MyNS
{
  class Point
  {
    int x, y;
  public:
    Point(int _x, int _y):
      x(_x), y(_y) {}
  };
}

int main()
{
  MyNS::Point p(10,20);
  return 0;
}
```

Понятие пакетов в Java

- Пакеты в Java предназначены:
  - *Для организации кода*

Имя пакета соответствует директории

Название пакета класса указывается отдельным оператором




```
package testaccess.MyPackege;

public class Point {
    public Point(int x, int y)
    …
}
```

  - *Для скрытия реализации*
    - Классы, поля и методы закрытые и с доступом по умолчанию недоступны для создания объектов вне пакета

Понятие пакетов в Java

- Пример использования класса из пакета в Java

Импорт позволяет использовать короткую запись создания объекта
* импортирует все классы из пакета

Создание объекта из пакета

Создание объекта из пакета с учетом импорта

```
package testaccess;

import testaccess.MyPackege.Point;
import testaccess.MyPackege.*;

public class Main {
  public static void main(String[] args){
    testaccess.MyPackege.Point a =
      new testaccess.MyPackege.Point(0,0);
    a.move(0, 0);

    Point b= new Point(0, 0);
    b.move(10, 20);
  }
}
```

МЕТОДЫ.
СПОСОБЫ ПЕРЕДАЧИ ДАННЫХ.

Способы передачи данных

- Какие способы передачи данных есть в C++?
  - **По значению**

```
void f(int a)  { a = 10; }
```

(копия передаваемого значения)

  - **По указателю**

```
void f(int *a) { *a = 10;}
```

(передача указателя по значению)

  - **По ссылке**

```
void f(int &a) { a = 10; }
```

(неявная передача указателя по значению)

- В чем разница между способами передачи данных?

Способы передачи данных в Java

- В Java «базовые типы данных» такие как int, double, boolean … ***передаются по значению***
- Все объекты и массивы ***создаются по ссылке***
  - Следующая запись создает лишь ссылку на объект

```
Point p;
```

  - По ссылке память не выделяется

```
Point p;
p.move(10, 20); //Исключительная ситуация!
```

  - Чтобы создать объект необходимо явно вызвать оператор ***new***

```
Point p = new Point();
p.move(10, 20); //Исключения нет
```

- Как следствие ***все объекты в Java передаются по ссылке!***

КОНСТРУКТОРЫ.

Конструкторы

- Конструктор – специальный метод, который автоматически вызывается при создании объекта
- Возможные функции конструктора в C++:
  - выделение памяти для полей класса
  - инициализация полей начальными значениями
  - создание объекта по образцу (копирование)
  - преобразование типа

Конструкторы

- Какие типы конструкторов существуют в C++?
  - Конструктор по умолчанию

```
Point():x(0), y(0) {}
```

  - Конструктор приведения типов

```
Point(int v):x(v), y(v) {}
```

  - Конструктор копирования

```
Point(const Point &p):x(p.x), y(p.y) {}
```

  - Инициализирующий конструктор

```
Point(int _x, int _y):x(_x), y(_y) {}
```

- Когда какие конструкторы вызываются?

Конструкторы

- Какие типы конструкторов существуют в Java?
  - Конструктор по умолчанию – реализовать возможно

```
class Point{
   private int x = 0, y = 0;
   private A a = new A(10, 20, 30);
   public Point(){}
```

  - Конструктор приведения типов – **не допустим по синтаксису языка, объекты создаются явным образом через new!**

Не допустима запись `Point p = 10;`

  - Конструктор копирования – **когда конструктор копирования нужен?**

(можно реализовать интерфейс Cloneable)

  - Инициализирующий конструктор – есть

```
Point(int _x, int _y){ x = _x; y = _y; }
```

Конструкторы

- Особенности реализации конструкторов
  - Если конструкторов нет, то при создании объектов используется конструктор по умолчанию
    - Конструктор по умолчанию реализует компилятор неявным образом
  - Если в классе реализован инициализирующий конструктор и не реализован конструктор по умолчанию, то при создании объекта нельзя использовать конструктор по умолчанию
    - Конструктор по умолчанию не реализуется компилятором

ПЕРЕГРУЗКА ОПЕРАЦИЙ.

О перегрузке операций

- В C++ можно перегрузить операции

```
class Point{
  int x, y;
public:
  ...
  Point operator + (Point const &obj) {
    return Point(x + obj.x, y + obj.y);
  }
};
```

- Перегрузка операций позволяет использовать более наглядную форму записи

```
Point a(1,2), b(3,4), c;
c = a + b; <==> c.operator = (a.operator + (b));
```

О перегрузке операций

- Что с перегрузкой операций в Java?
- **В Java перегрузка операций не доступна**
- Некоторые предпосылки:
  - Нужен ли в Java **operator +** ?
    - Возможно
  - Нужен ли в Java **operator =** ?
    - Нет. В Java объекты не создаются и не возвращаются по значению
  - Нужен ли в Java **operator ==** ?
    - Что будет если написать **`if(a == b){…}`**?
    - В Java даже в стандартных структурах данных чаще используют **a.equals(b)**
  - Нужен ли в Java **operator ->** ?
    - Нет. Нет такой операции
  - Нужен ли в Java **operator <<** ?
    - Нет. В Java используется другой механизм (далее пример)

О перегрузке операции вывода

- Вывод объекта в C++ через поток вывода

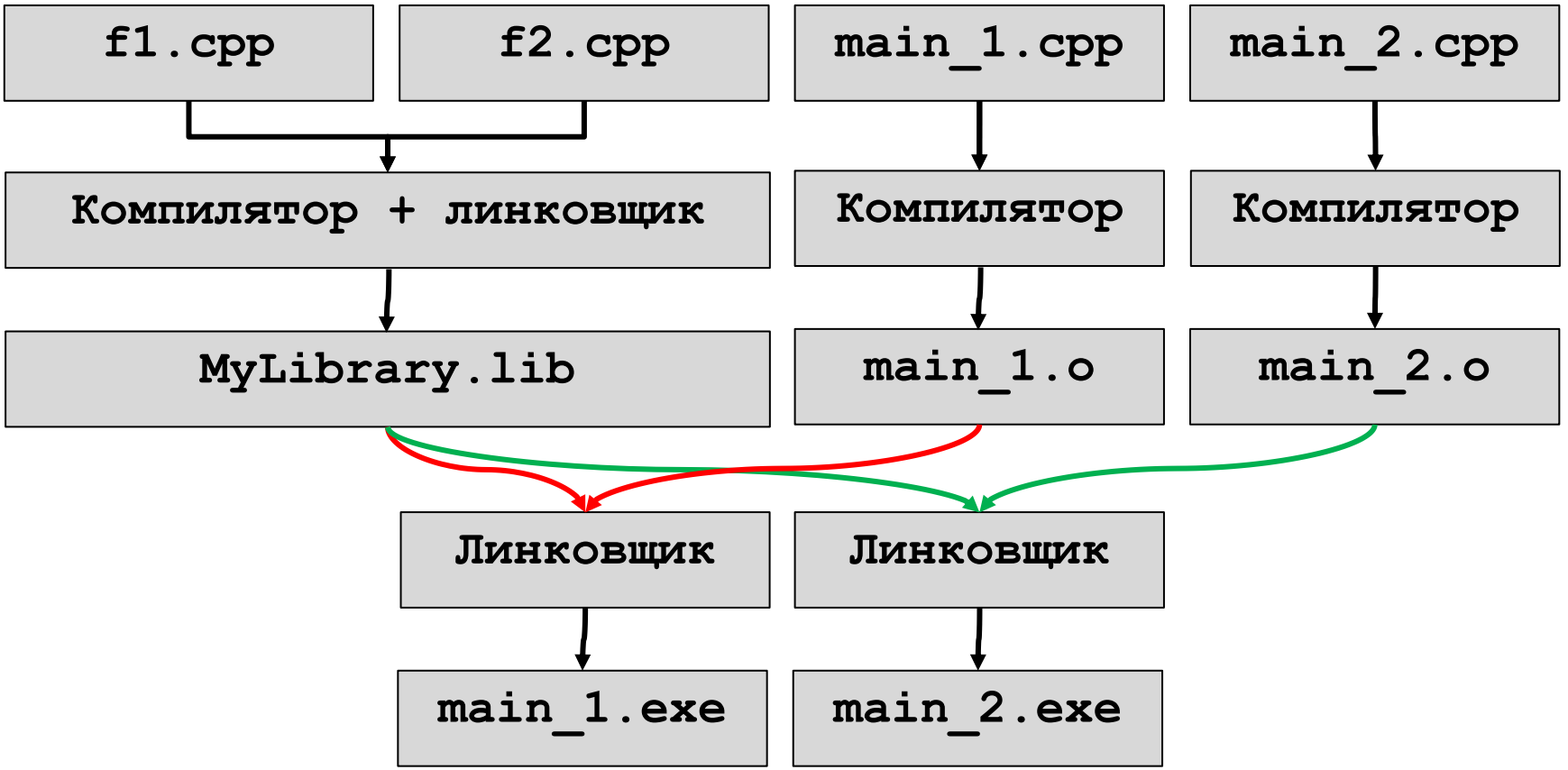
```
ostream& operator<< (ostream &out, const Point &point) {
    out << "(" << point.x << ", " << point.y << ")";
    return out;
}
Point p(10, 20);
cout << p;
```

- Вывод в Java (все классы наследники Object, с мет. `toString`)

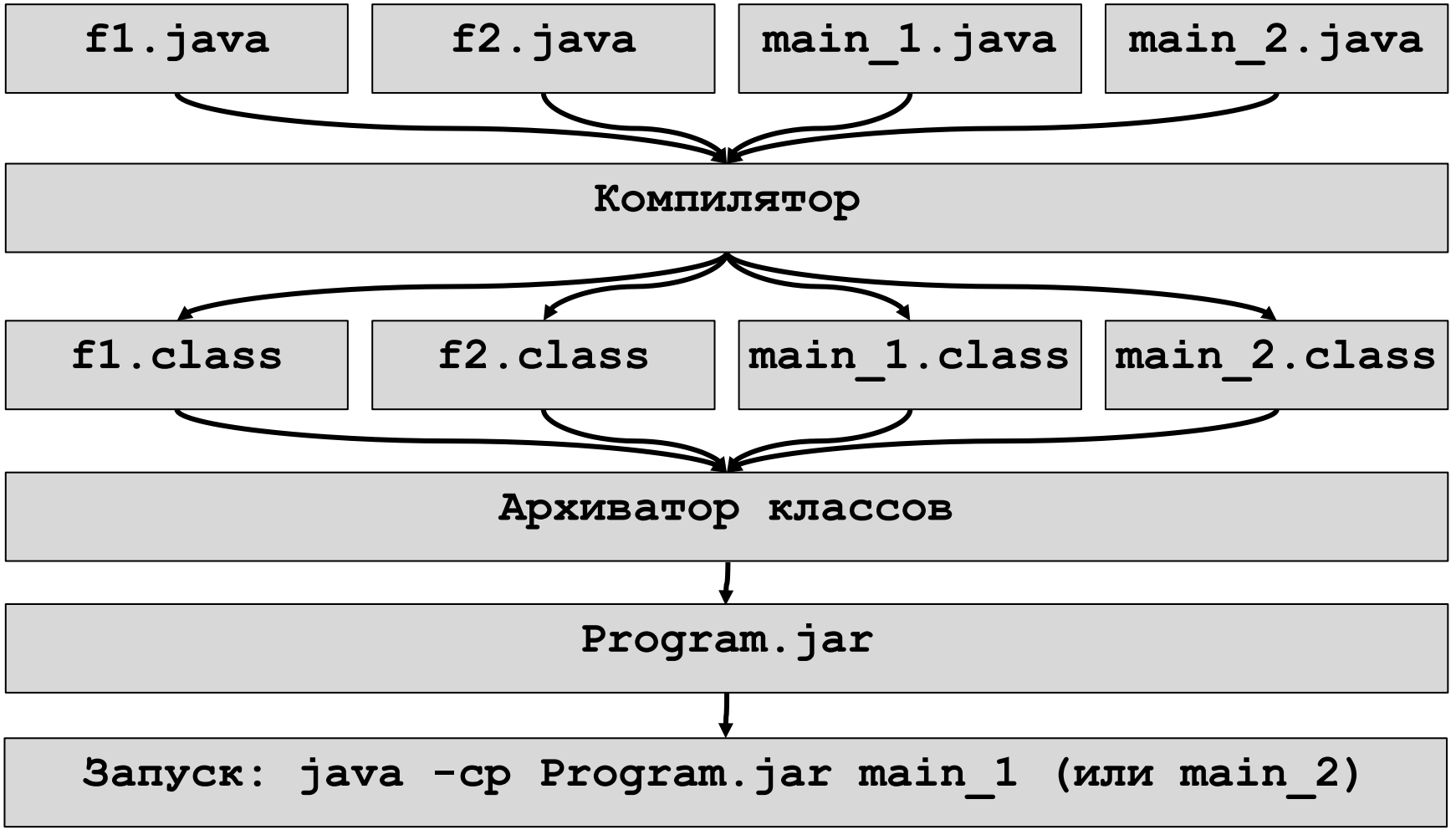
```
public class Point {
    @Override
    public String toString() { return "("+x+","+y+")"; }
}
Point p = new Point(10, 20);
System.out.println(p);
```

JAR АРХИВЫ

Принцип разработки программ в C++

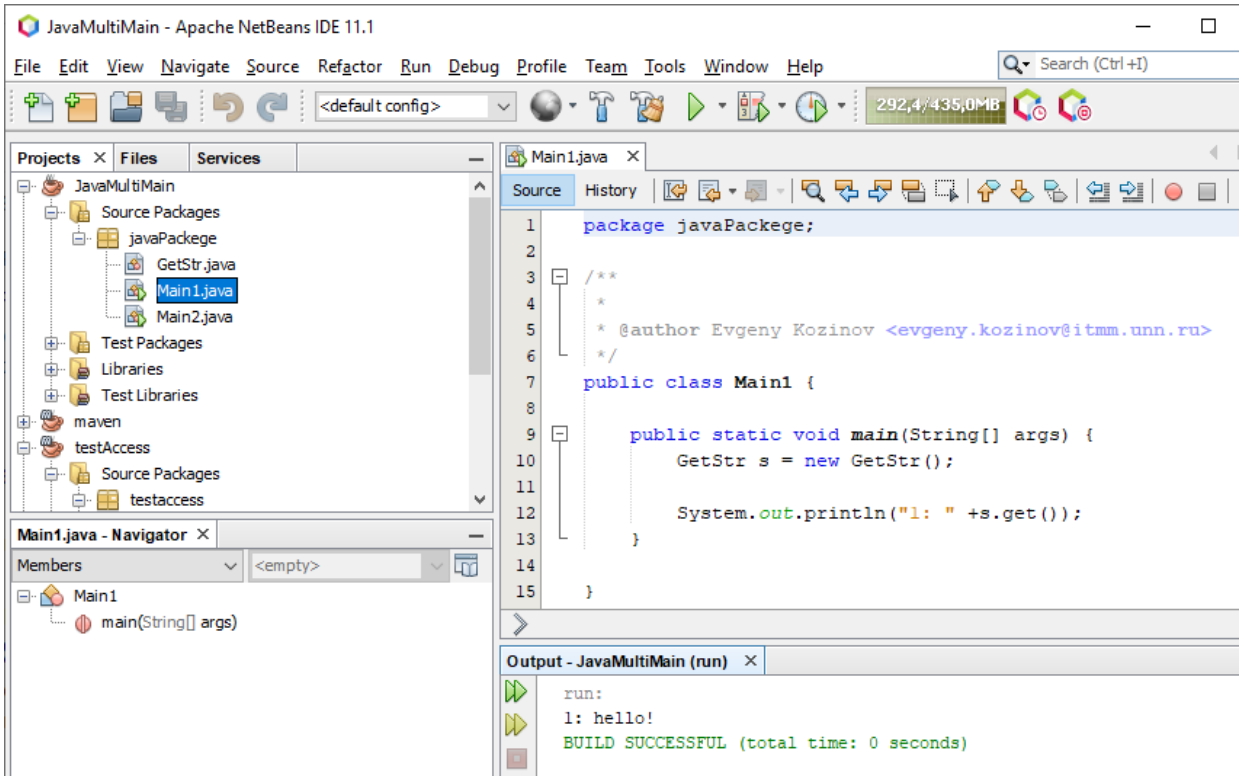


Принцип разработки программ в Java



Jar архивы

- Jar архивы это ***не исполняемые файлы!***
- Jar архивы содержат
  - Все скомпилированные классы в виде файлов
  - Метаданные – версия компилятора, версия java, какой класс использовать при запуске по умолчанию …
- Jar архивы позволяют ***повысить безопасность кода!***
  - В JDK есть механизм подписания Jar архивов и методы дальнейшей проверки подписи
  - Подписи помогают отследить вмешательство в скомпилированные классы



ДЕМОНСТРАЦИЯ ИСПОЛЬЗОВАНИЯ JAR АРХИВОВ

ЗАКЛЮЧЕНИЕ

Заключение

- В лекции рассмотрены основные понятия связанные с классами и объектами
- В лекции рассмотрены основные отличия синтаксиса написания классов C++ от классов Java
- В лекции показывается способ создания и использования Jar архивов
- Для прохождения практики рекомендуется
  - Установить JDK
  - Установить одну из сред разработки
    - NetBeans https://netbeans.org/
    - IntelliJ IDEA https://www.jetbrains.com/ru-ru/idea/

Литература

1. Программирование на Java - http://www.intuit.ru/studies/courses/16/16/info
2. Построение распределенных систем на Java - http://www.intuit.ru/studies/courses/633/489/info
3. Дистрибутивы средств разработки ПО - http://www.oracle.com/technetwork/java/javase/downloads/index.html
4. Официальная документация по языку программирования Java - https://docs.oracle.com/javase/tutorial/

Контакты

Нижегородский государственный университет
http://www.unn.ru

Институт информационных технологий, математики и механики
http://www.itmm.unn.ru

Козинов Е.А.
evgeny.kozinov@itmm.unn.ru