

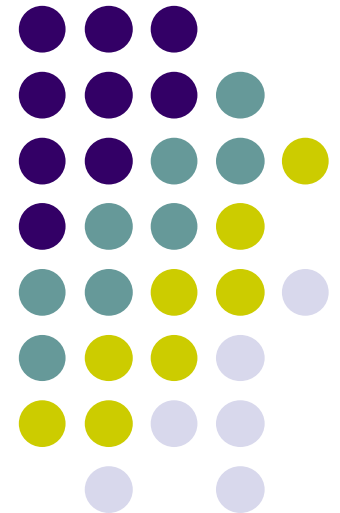
Распределенные системы

Практическая часть 1

Технологии распределенного программирования

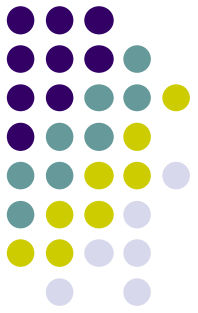
к.т.н. Приходько Т.А.

Кубанский государственный университет
кафедра вычислительных технологий



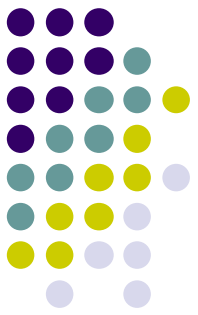
План

- Средства автоматического распараллеливания программ
- MPI
- Настройка рабочего места для выполнения практических заданий.
- Выполнение первой программы



СРЕДСТВА АВТОМАТИЧЕСКОГО РАСПАРАЛЛЕЛИВАНИЯ ПРОГРАММ

Варианты архитектур

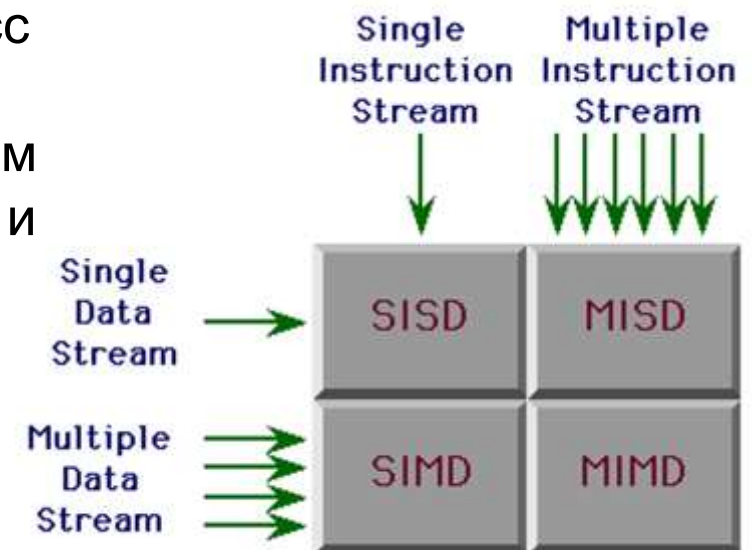


Чтобы процессоры могли считаться автономными, они должны, по меньшей мере, обладать **собственным независимым управлением**.

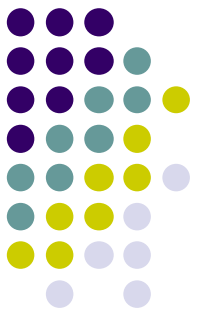
- По этой причине параллельный компьютер, архитектура которого устроена по схеме "одна команда для многих данных" (англ. Single Instruction - Multiple Data, SIMD), не может считаться распределенной системой.

Под **независимостью процессов** подразумевается тот факт, что каждый процесс имеет свое собственное состояние, представляемое набором данных, включающим текущие значения счетчика команд, регистров и переменных, к которым процесс может обращаться и которые может изменять. Состояние каждого процесса является полностью закрытым для других процессов: другие процессы не имеют к нему прямого доступа и не могут изменять его.

Классификация Флина



Средства автоматического распараллеливания

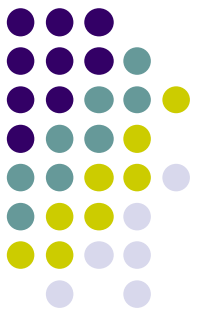


- Средства автоматического распараллеливания – наиболее быстрый способ получить параллельную программу из последовательной, но **степень параллелизма кодов, полученных автоматически, ниже степени параллелизма кодов программ, в которых параллелизм закладывается программистом.** Так или иначе, но машина предпочтет не распараллеливать любой подозрительный фрагмент программы, в то время, как программист знает, какая часть алгоритма, не являющаяся заведомо параллельной, тем не менее может быть распараллелена.

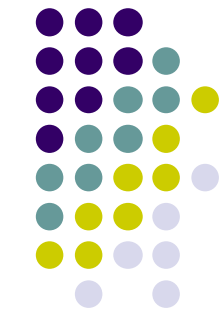
Способы синхронизации параллельного взаимодействия:

- Через разделяемую память
- Путем обмена сообщениями

Некоторые программные инструменты параллелизма

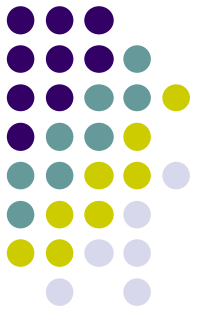


- [OpenMP](#) — стандарт интерфейса приложений для параллельных систем с общей памятью.
- [POSIX Threads](#) — стандарт реализации потоков (нитей) выполнения.
- [Windows API](#) — многопоточные приложения для C++.
- [PVM \(Parallel Virtual Machine\)](#) позволяет объединить разнородный (но связанный сетью) набор компьютеров в общий вычислительный ресурс.
- [MPI \(Message Passing Interface\)](#) — стандарт систем передачи сообщений между параллельно исполняемыми процессами, ориентирован на системы с распределенной памятью.



MPI

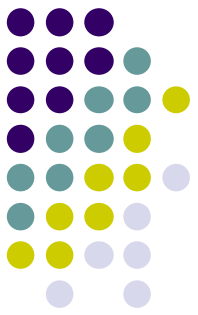
MPI



- MPI (Message Passing Interface) — интерфейс обмена сообщениями (информацией) между одновременно работающими вычислительными процессами. Он широко используется для создания параллельных программ для вычислительных систем с распределённой памятью (кластеров).
- MPI— это не язык, это библиотека подпрограмм обмена сообщениями - спецификация разработанная в 1993—1994 годах группой MPI Forum, в состав которой входили представители академических и промышленных кругов. Он стал первым стандартом систем передачи сообщений.

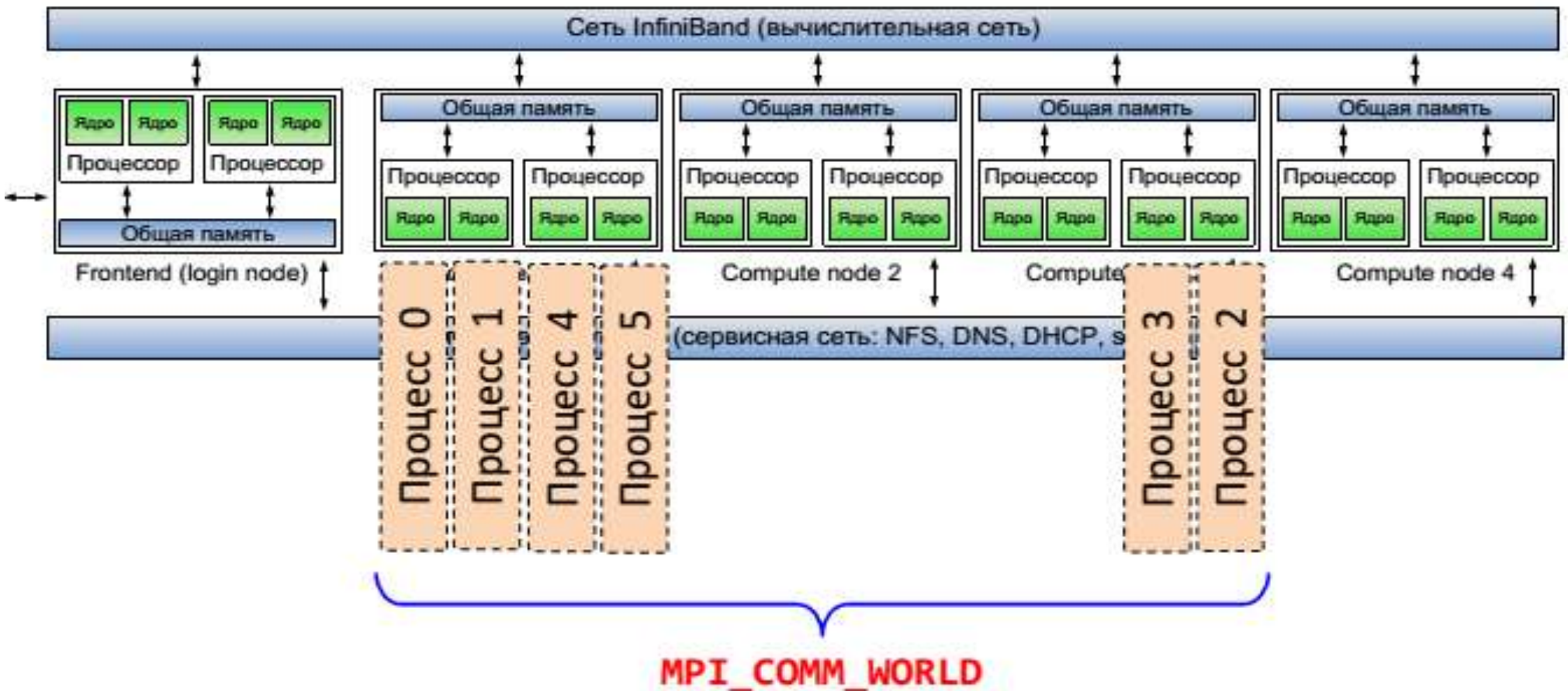
Официальный сайт MPI: <http://www.mpi-forum.org>

MPI



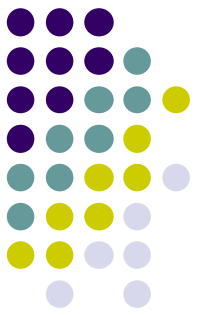
- Для MPI принято писать программу, содержащую код всех ветвей сразу. MPI-загрузчиком запускается указываемое количество экземпляров программы.
- Каждый экземпляр определяет свой порядковый номер в запущенном коллективе и, в зависимости от этого номера и размера коллектива, выполняет ту или иную ветку алгоритма.
- Такая модель параллелизма называется *Single program/Multiple data* (SPMD) и является частным случаем модели *Multiple instruction/Multiple data* (MIMD).
- Каждая ветвь имеет пространство данных, полностью изолированное от других ветвей. Обмениваются данными ветви только в виде сообщений MPI.

Стандарт MPI



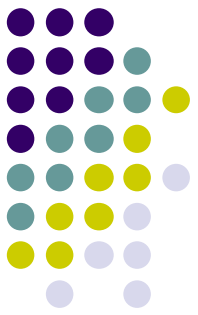
- **Коммуникатор (communicator)** – множество процессов, образующих логическую область для выполнения коллективных операций (обменов информацией и др.)
 - В рамках коммуникатора ветви имеют номера: $0, 1, \dots, n - 1$

MPI



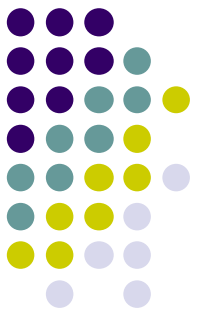
- Все ветви запускаются загрузчиком одновременно как процессы Unix. Количество ветвей фиксировано - в ходе работы порождение новых ветвей невозможно.
- Если MPI-приложение запускается в сети, запускаемый файл приложения должен быть построен на каждой машине.
- Загрузчик MPI приложений - утилита `mpirun`. Параллельное приложение будет образовано N задачами-копиями. В момент запуска все задачи одинаковы, но получают от MPI разные номера от 0 до $N-1$. В тексте параллельной программы эти номера используются для указания конкретному процессу, какую ветвь алгоритма он должен выполнять.

MPI (проблемы при использовании)



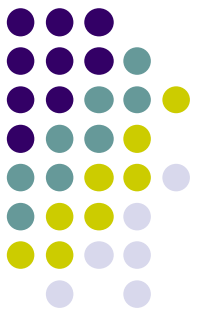
- во-первых, перед запуском приложения необходимо копирование приложения на все компьютеры кластера;
- во-вторых, перед запуском приложения необходима информация о реально работающих компьютерах кластера для редактирования **файла конфигурации кластера**, который содержит имена машин;
- в-третьих, в MPI существующей реализации не поддерживается **динамическое регулирование процессов**, т.е. если один из узлов выходит из строя, то общий вычислительный процесс прекращается, и если добавляются новые узлы, то в запущенном ранее процессе они не участвуют.

MPI (этапы разработки программы)



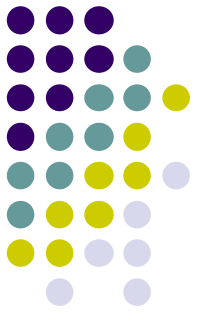
- Создание параллельной программы можно разбить на следующие этапы:
 - последовательный алгоритм подвергается декомпозиции (распараллеливанию), т.е. разбивается на независимо работающие ветви;
 - для взаимодействия в ветви вводятся две нематематические функции: прием и передача данных;
 - распараллеленный алгоритм записывается в виде программы, в которой операции приема и передачи записываются в терминах MPI, тем самым обеспечивая связь между ветвями.

Сравнение MPI с другими средствами



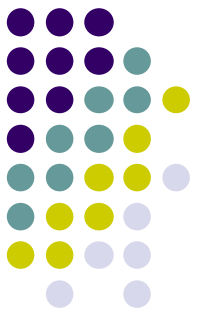
	MPI	Threads	OpenMP*
Portable	✓		✓
Scalable	✓	✓	✓
Performance Oriented	✓		✓
Supports Data Parallel	✓	✓	✓
Incremental Parallelism			✓
High Level			✓
Serial Code Intact			✓
Verifiable Correctness			✓
Distributed Memory	✓		

Версии MPI



- Версия MPI-1 вышла в 1994 году
- Версия MPI-2 вышла в 1998 году, первая реализация появилась в 2002 году.
- Версия MPI-2.1 вышла в начале сентября 2008 года
- Версия MPI 2.2 вышла 4 сентября 2009 года.
- Версия MPI 3.0 вышла 21 сентября 2012 года.

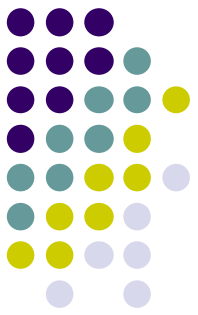
Спецификации MPI



Спецификация MPI-1

- Содержит описание стандарта программного интерфейса обмена сообщениями. Спецификация учитывает опыт предшествующих разработок и ориентирована на большую часть аппаратных платформ. Несмотря на то, что MPI рассчитано на использование с языками C/C++ и Fortran, семантика в значительной степени не зависит от языка.
- В MPI-1 описываются интерфейсы процедур двухточечного и коллективного обмена, сбора информации, организации обменов в группах процессов, синхронизации процессов, виртуальные топологии, привязки к языкам программирования и т. д.

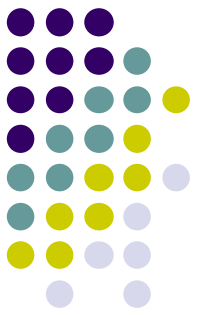
Спецификации MPI



Спецификация MPI-2

Является дальнейшим развитием MPI. Новое в MPI-2:

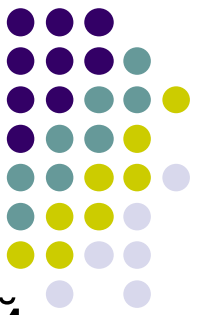
- возможность создания новых процессов во время выполнения MPI- программы (в MPI-1 количество процессов фиксировано, крах одного приводит к краху всей программы);
- новые разновидности двухточечных обменов (односторонние обмены);
- новые возможности коллективных обменов;
- поддержка внешних интерфейсов;
- операции параллельного ввода-вывода с файлами.



Реализации MPI

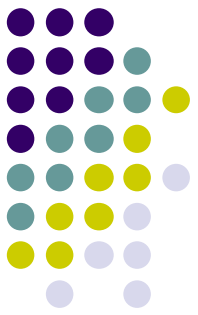
- **MPICH2** (Open source, Argone NL)
<http://www.mcs.anl.gov/research/projects/mpich2>
- MVAPICH2
- IBM MPI
- Cray MPI
- Intel MPI
- HP MPI
- SiCortex MPI
- Open MPI (Open source, BSD License)
<http://www.open-mpi.org>
- Oracle MPI
- [MPJ Express](#) — MPI на Java
- [WMPI](#) WMPI — реализация MPI для [Windows](#)
- ...

Реализации MPI



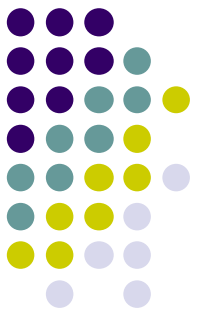
- **MPI CHameleon (MPICH)** является свободно распространяемой “opensource” реализацией MPI. Этот пакет доступен в исходных кодах, поэтому допускает гибкую настройку.
- Поддерживается работа в различных версиях ОС UNIX, Mac OS и в последних версиях Microsoft Windows. В последнем случае имеется возможность установки из бинарных файлов.
- MPICH соответствует спецификации MPI-2.
- Поддерживаются различные коммуникационные среды (в т.ч. 10 Gigabit Ethernet, InfiniBand, Myrinet, Quadrics).
- Пока не поддерживаются системы, гетерогенные по форматам хранения данных. Имеется версия с поддержкой пакета Globus.
- **Официальный сайт MPICH**
- <http://www-unix.mcs.anl.gov>

Реализации MPI



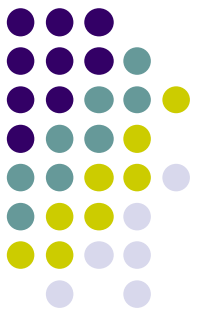
- «Производные» от MPICH
- **LAM/MPI** (Университет шт. Индиана.)
- **MPICH GM** (Myricom) - MPICH с поддержкой среды Myrinet.
- **MVAPICH** (Университет шт. Огайо) — с поддержкой среды Infiniband.
- **MPI/Pro** (MPI Software Technology).
- **Scali MPI Connect.**
- **Intel® MPI** входит в состав Intel® Cluster Toolkit. Это коммерческая реализация MPI, оптимизированная для архитектуры Intel. **Сайт в Интернете:** <http://www.intel.com>

Реализации MPI



- **OpenMPI** - “opensource” реализация MPI-2, разрабатываемая консорциумом представителей академических, научных и индустриальных кругов.
- Полное соответствие спецификации MPI-2. Поддержка различных ОС.
- Поддержка различных коммуникационных сред.
- **Сайт в Интернете:** <http://www.open-mpi.org>

Реализации MPI

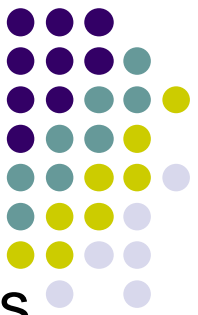


- **Microsoft MPI (MS-MPI v7.1)** входит в состав Compute Cluster Pack SDK.
- Ориентирован на работу в среде ОС Microsoft Windows и доступен, в том числе, по лицензии MSDN Academic Alliance. Входит в состав Microsoft HPC Server 2012.
- Основан на MPICH2, но включает дополнительные средства управления заданиями.
- Поддерживается спецификация MPI-2:

www.microsoft.com/en-us/download

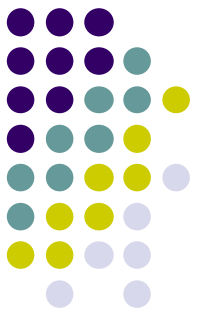
Согласно документации поддерживаются Windows 7, Windows 8,10

Реализации MPJ



- Most of those early projects are no longer active. **mpiJava** is still used and supported. We will describe it in detail later.
- Other more contemporary projects include
 - **MPJ**
 - An API specification by the “Message-passing Working Group” of the *Java Grande Forum*. Published 2000.
 - **CCJ**
 - An MPI-like API from some members of the Manta team. Published 2002.
 - **MPJava**
 - A high performance Java message-passing framework using **java.nio**. Published 2003.
 - **JMPI**
 - An implementation of the MPJ spec from University of Massachusetts. Published 2002.
 - **JOPI**
 - Another Java Object-Passing Interface from U. Nebraska-Lincoln. Published 2002.

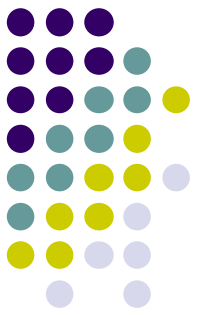
Реализации MPJ



- Один из ранних и наиболее живучих проектов - **mpiJava** все еще поддерживается и широко используется.
- Другие более современные проекты:
 - **MPJ**
 - An API specification by the “Message-passing Working Group” of the *Java Grande Forum*. Published 2000.
 - **CCJ**
 - An MPI-like API from some members of the Manta team. Published 2002.
 - **MPJava**
 - A high performance Java message-passing framework using java.nio. Published 2003.
 - **JMPI**
 - An implementation of the MPJ spec from University of Massachusetts. Published 2002.
 - **JOPI**
 - Another Java Object-Passing Interface from U. Nebraska-Lincoln. Published 2002.



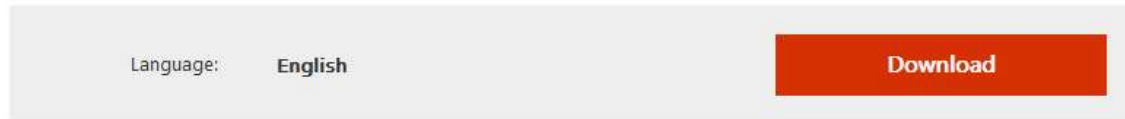
НАСТРОЙКА РАБОЧЕГО МЕСТА



Установка MS-MPI

- <https://www.microsoft.com/en-us/download/details.aspx?id=49926>
- скачиваем **msi**-файл (размер около 2 Мбайт):

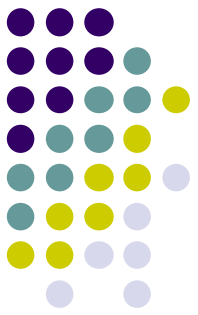
Microsoft MPI v7 (Archived)



Stand-alone, redistributable and SDK installers for Microsoft MPI. This version of MS-MPI is now archived. The link to the latest version is available in the Details section below.

- Для установки потребуются права Администратора системы (запуск MPI-программ выполняется соответствующей службой).

- Можно использовать MPICH2 для MS Windows, скачать по адресу:
<http://www.mpich.org/downloads/>



Установка MS-MPI

- Запускаем скачанный файл, он нам устанавливает MPI SDK:



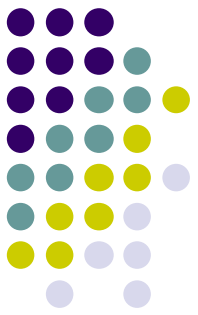
Welcome to the Microsoft MPI SDK
(7.0.12437.6) Setup Wizard

The Setup Wizard will install Microsoft MPI SDK (7.0.12437.6)
on your computer. Click Next to continue or Cancel to exit
the Setup Wizard.

Можно попробовать эту инструкцию для дальнейшей настройки:


<http://www.math.ucla.edu/~mputhawala/PPT.pdf>

Установка MPJ



1. Загружаем MPJ Express в виде zip – файла отсюда:

<http://sourceforge.net/projects/mpjexpress/files/releases> и распаковываем.









MPJ Express: Parallel Computing for Java

Brought to you by: [aamirshafi](#), [bryancarpenter](#), [hamza100](#), [mohsanjameel](#), [umarbt](#)

Summary | **Files** | Reviews | Support | Wiki | Mailing Lists

Looking for the latest version? [Download mpj-v0_44.zip \(8.2 MB\)](#)

[Home](#) / [releases](#)

Name ▾	Modified ▾	Size ▾	Downloads / Week ▾
↑ Parent folder			
mpj-v0_44.zip	2015-04-17	8.2 MB	43  
mpj-v0_44.tar.gz	2015-04-17	7.5 MB	17  
mpj-v0_43.zip	2014-07-21	7.6 MB	1  

It's

Wh

☐ I

☐ I


☐ F


Настройка системных переменных




1. Входим в «Дополнительные параметры системы»

Панель управления —
домашняя страница

 Диспетчер устройств

 Настройка удаленного
доступа

 Защита системы

 **Дополнительные параметры
системы**

Просмотр основных сведений о вашем компьютере

Выпуск Windows

Windows 10 Корпоративная LTSC

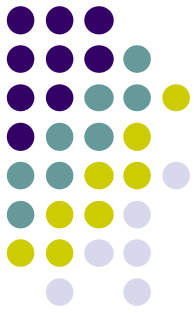
© Корпорация Майкрософт (Microsoft Corporation), 2018. Все пра

Система

Панель задач

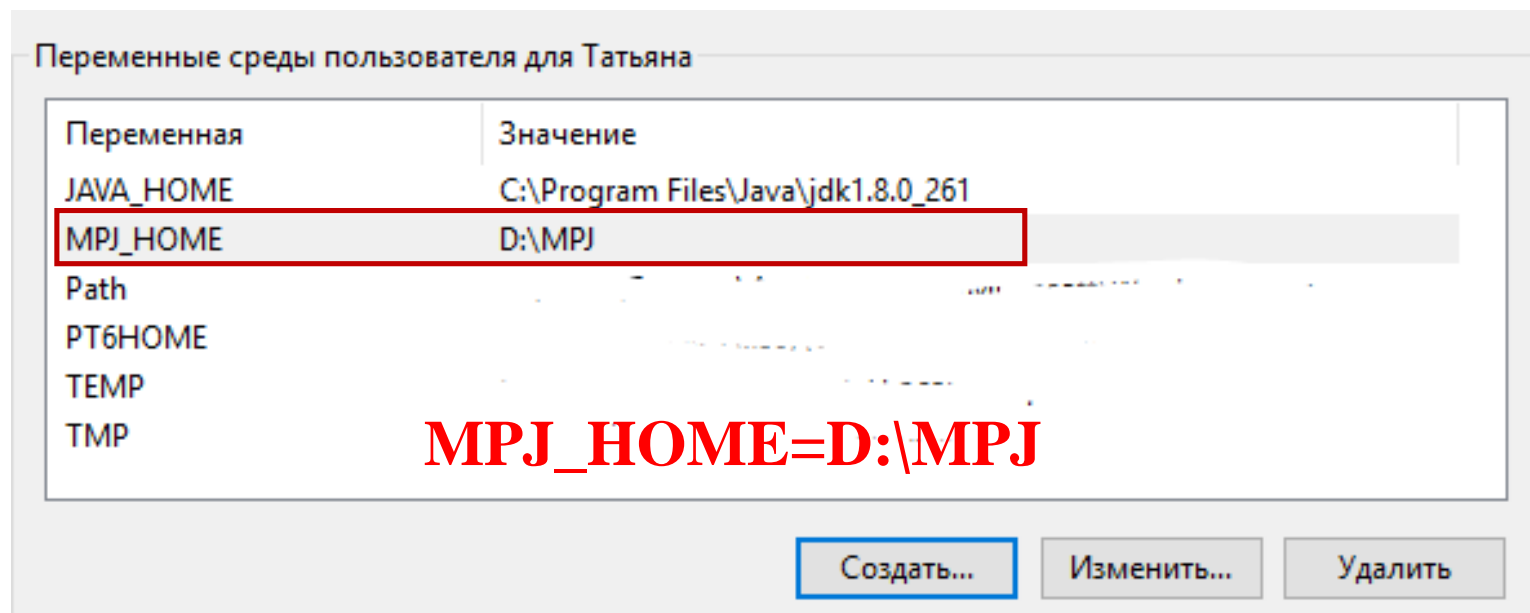
10.0.17134.1 (Win10.0.17134.1) @ 2.100

Установка MPJ



2. Извлекаем архив в выбранную вами директорию (*мы назовем ее условно "mpj directory"*). Теперь в вашей директории *mpj directory* вложена директория "mpj-v0_44" (возможно более новая).

- (2.1) Создаем переменную среды пользователя а MPJ_HOME:



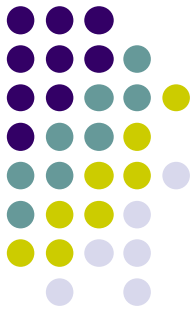
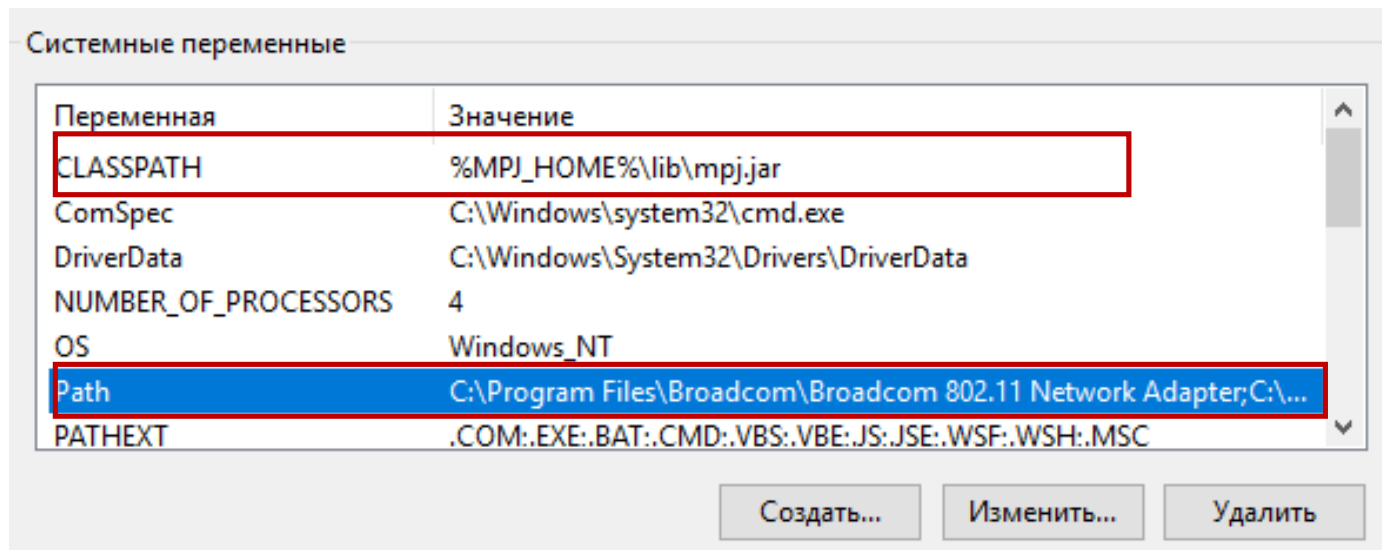
записываем туда путь к вашей директории *"mpj directory"*

\$MPJ_HOME=[mpj directory;] (без квадратных скобок)

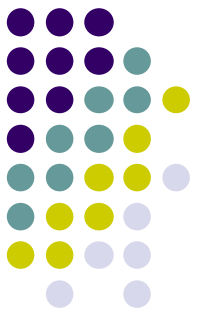
JAVA-HOME тоже пригодится – в ней путь к JDK

Установка MPJ

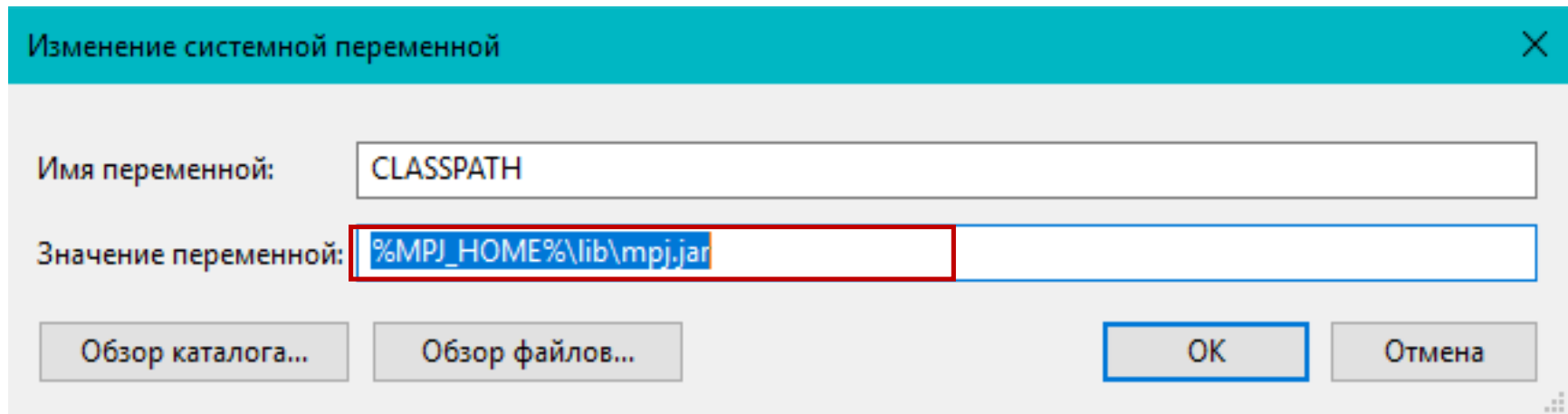
- (2.2) Добавляем MPJ bin директорию к переменной окружения PATH: **Path=\$PATH:\$MPJ_HOME/bin**



Установка MPJ



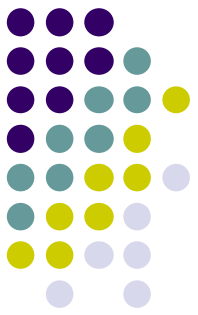
- (2.3) Добавьте к classpath:
CLASSPATH=.:\$MPJ_HOME/lib/mpj.jar



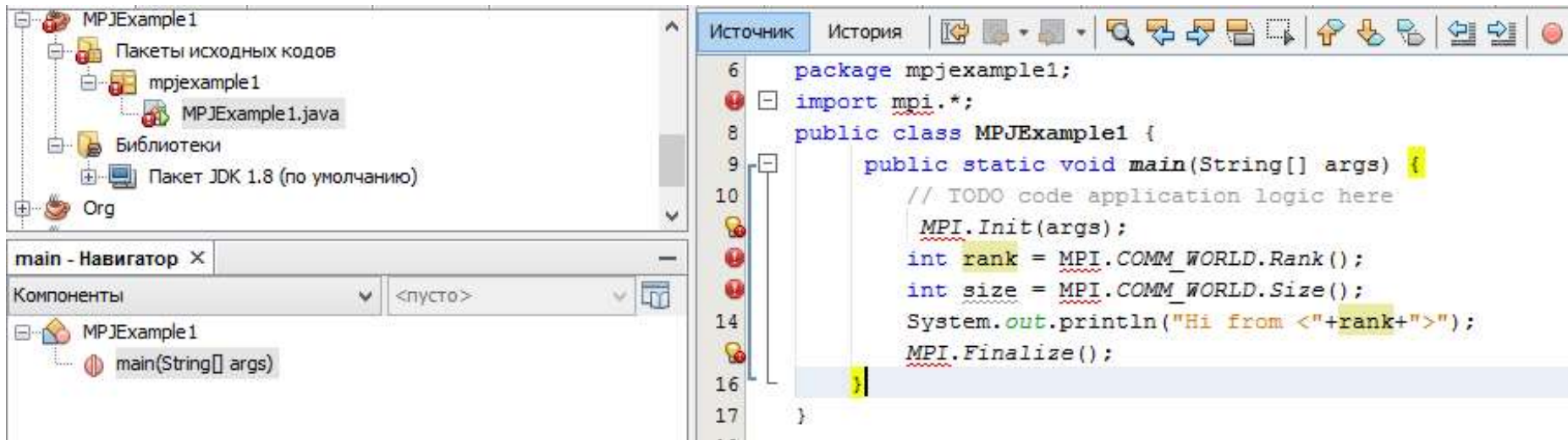
- Изменения системных переменных вступят в силу после перезагрузки, иначе получите ошибку:

```
[MPJRun.java]:[MPJRun.java]:MPJ_HOME environment found..  
java.lang.Exception: [MPJRun.java]:MPJ_HOME environment found..  
    at runtime.starter.MPJRun.<init>(MPJRun.java:155)  
    at runtime.starter.MPJRun.main(MPJRun.java:1238)
```

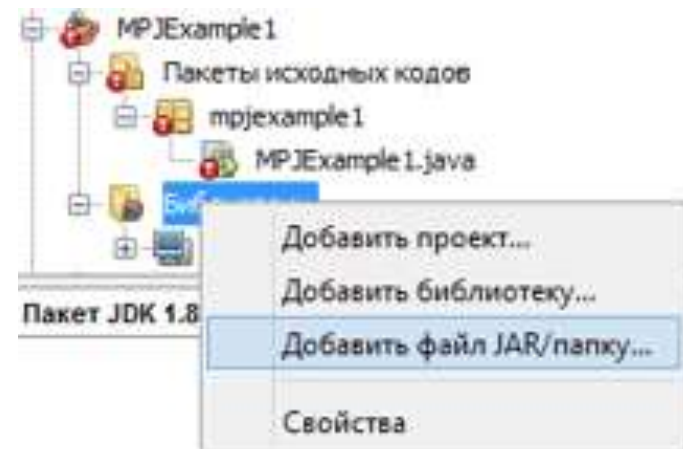

Настройка проекта в JavaBeans



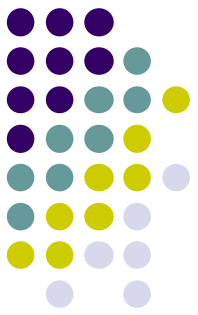
- Создаем новый JAVA-проект и помещаем в него код:



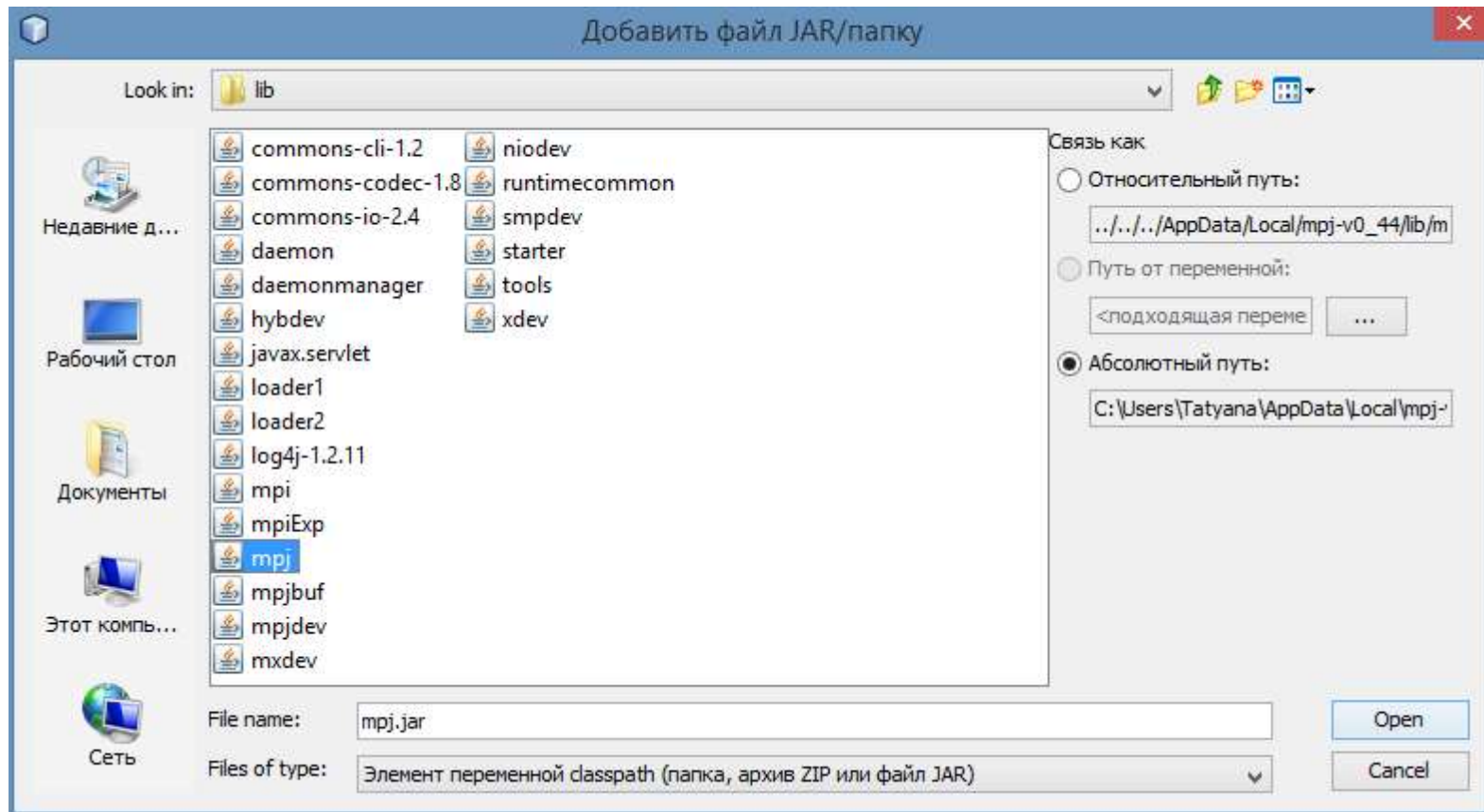
Библиотека не видна. Добавим ее:



Настройка проекта в JavaBeans

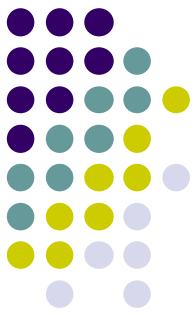


- Добавить JAR-файл mpj.jar к библиотеке проекта

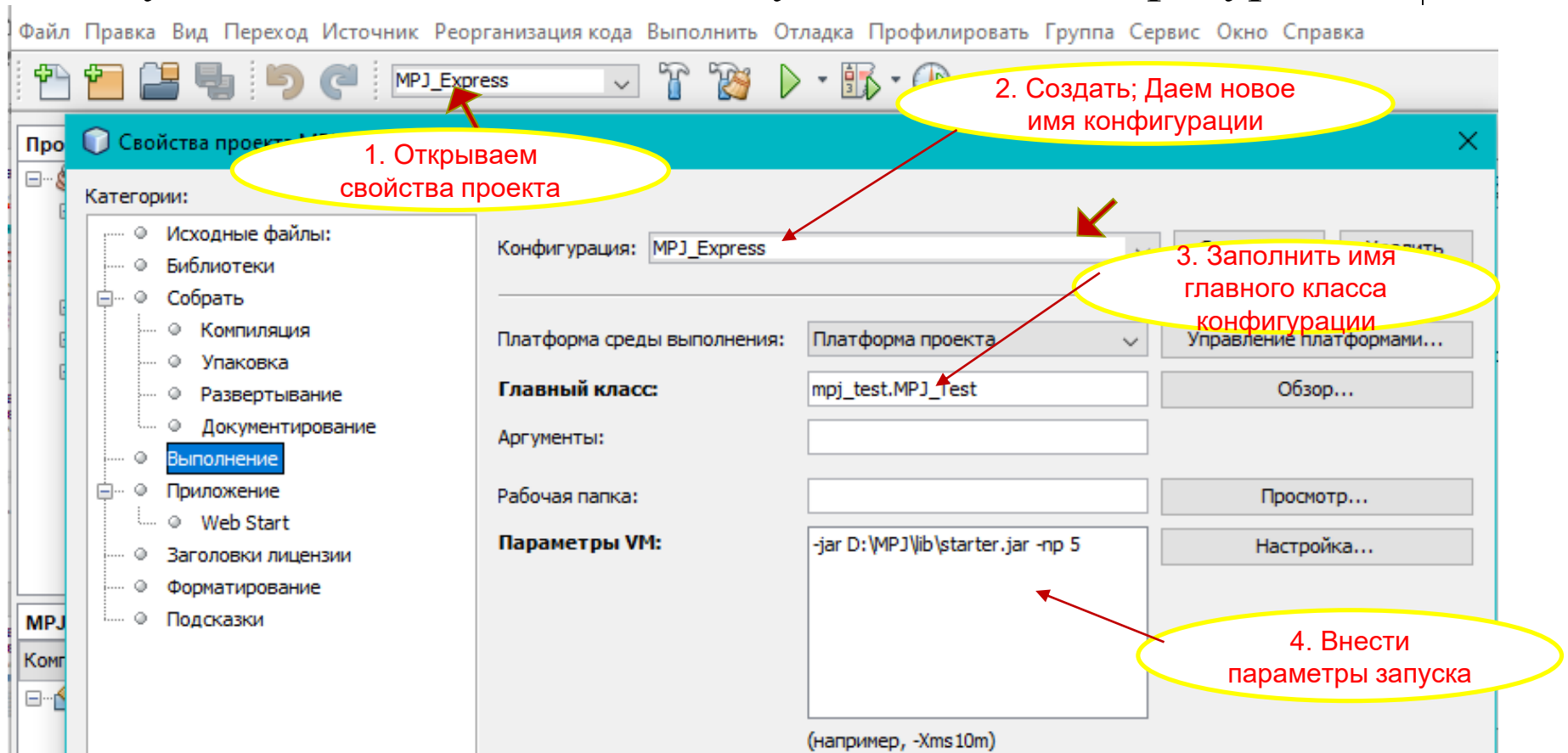


После этого библиотека будет видна. Ошибки в проекте исчезнут.

Настройка проекта в JavaBeans

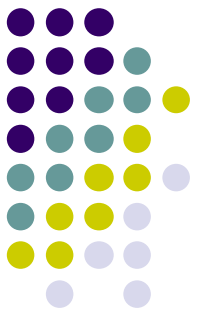


- Следующим шагом создаем новую Runtime конфигурацию:



Параметры запуска (5 – число процессов):

`-jar D:\...\MPJ\lib\starter.jar -np 5`



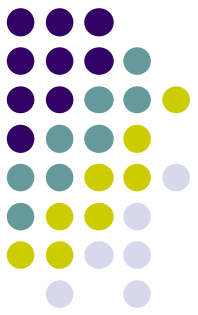
MPJ “Hello, World”

- Запускаем:

```
package mpjexample1;
import mpi.*;
public class MPJExample1 {
    public static void main(String[] args) throws Exception {
        MPI.Init(args);
        int rank = MPI.COMM_WORLD.Rank();
        int size = MPI.COMM_WORLD.Size();
        System.out.println("Hi from <"+rank+">");
        MPI.Finalize();
    }
}
```

- Получаем:

```
MPJ Express (0.44) is started in the multicore configuration
Hi from <3>
Hi from <1>
Hi from <2>
Hi from <4>
Hi from <0>
```



MPJ Режим отладки

- Вводим дополнительные параметры настройки конфигурации:

Конфигурация:

Платформа среды выполнения:

Главный класс:

Аргументы:

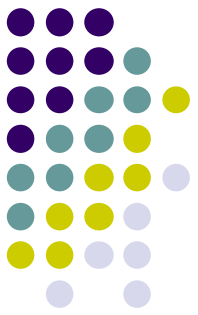
Рабочая папка:

Параметры VM:

(например, -Xms10m)

Дополнительные параметры командной строки для отладчика

- “-agentlib:jdwp=transport=dt_socket,server=y,suspend=y,address=8000”
- Отладчик подключится к порту, указанному в этой строке— значение по умолчанию равно 8000 и может быть изменено.



MPJ Режим отладки

- Устанавливаем точку останова и запускаем:

```
import mpi.*;
public class DebuggerDemo {
    public static void main(String[] args) {
        MPI.Init(args);
        int rank = MPI.COMM_WORLD.Rank();
        int size = MPI.COMM_WORLD.Size();
        System.out.println("Hello I am <"+rank+"> Total are <"+size+">");
        MPI.Finalize();
    }
}
```

6:1 INS

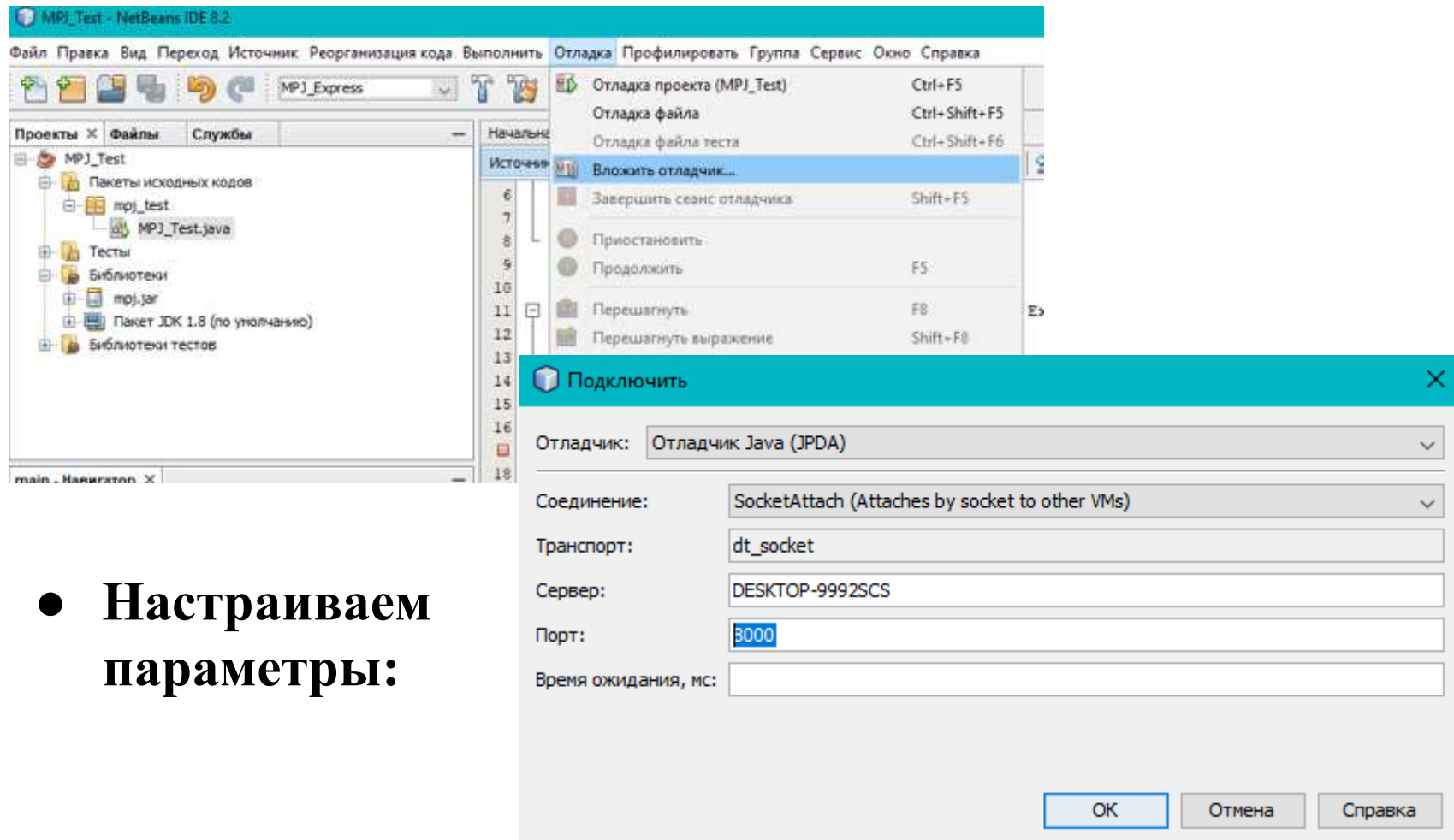
Output - MPJExpressDebuggerDemo (run) Tasks

run:
MPJ Express (0.36) is started in the multicore configuration
Listening for transport dt_socket at address: 8000

- Видим, что идет прослушивание сокета и порта 8000.

MPJ Режим отладки

- Подключаем отладчик:



- Настраиваем параметры:

MPJ Режим отладки

- Процесс отладки:

Нажимая на кнопки отладки можем контролировать каждый процесс отдельно

Файл Правка Вид Переход Источник Реорганизация кода Выполнить Отладка Профилировать Группа Сервис Окно Справка

MPJ_Express

285,6/400,0MB

MPJ_Test.java

```
12
13 MPI.Init(args);
14 int rank = MPI.COMM_WORLD.Rank();
15
16 int size = MPI.COMM_WORLD.Size();
17 System.out.println("Hi from <"+rank+">");
18 MPI.Finalize();
19 }
20
21
22
```

main - Навигатор

Компоненты

MPJ_Test

main(String[] args)

Наблюдения

Имя	Тип	Значение
До вызова 'println'		
Аргументы		
"Hi from <"+rank+">"	String	"Hi from <4>"
История возвращенных значений		
return Size()	int	5
Статически		
args	String[]	#633(length=3)
rank	int	4
size	int	5

Выбор контролируемого процесса

MPJ Режим отладки

- Процесс отладки:

The screenshot displays the MPJ Express IDE interface during a debugging session. The top toolbar contains various icons for file operations, execution, and debugging. The 'Проекты' (Projects) tab is active, showing a tree view of the project structure. The 'MPJ_Test.main:18' method is selected. The 'Источник' (Source) tab shows the code for 'MPJ_Test.java'. The code is as follows:

```
12
13  MPI.Init(args);
14  int rank = MPI.COMM_WORLD.Rank();
15
16  int size = MPI.COMM_WORLD.Size();
17  System.out.println("Hi from <"+rank+">");
18  MPI.Finalize();
19
20
21
22
```

The 'Наблюдения' (Observations) tab is active, showing the output of the 'main' method. The output is as follows:

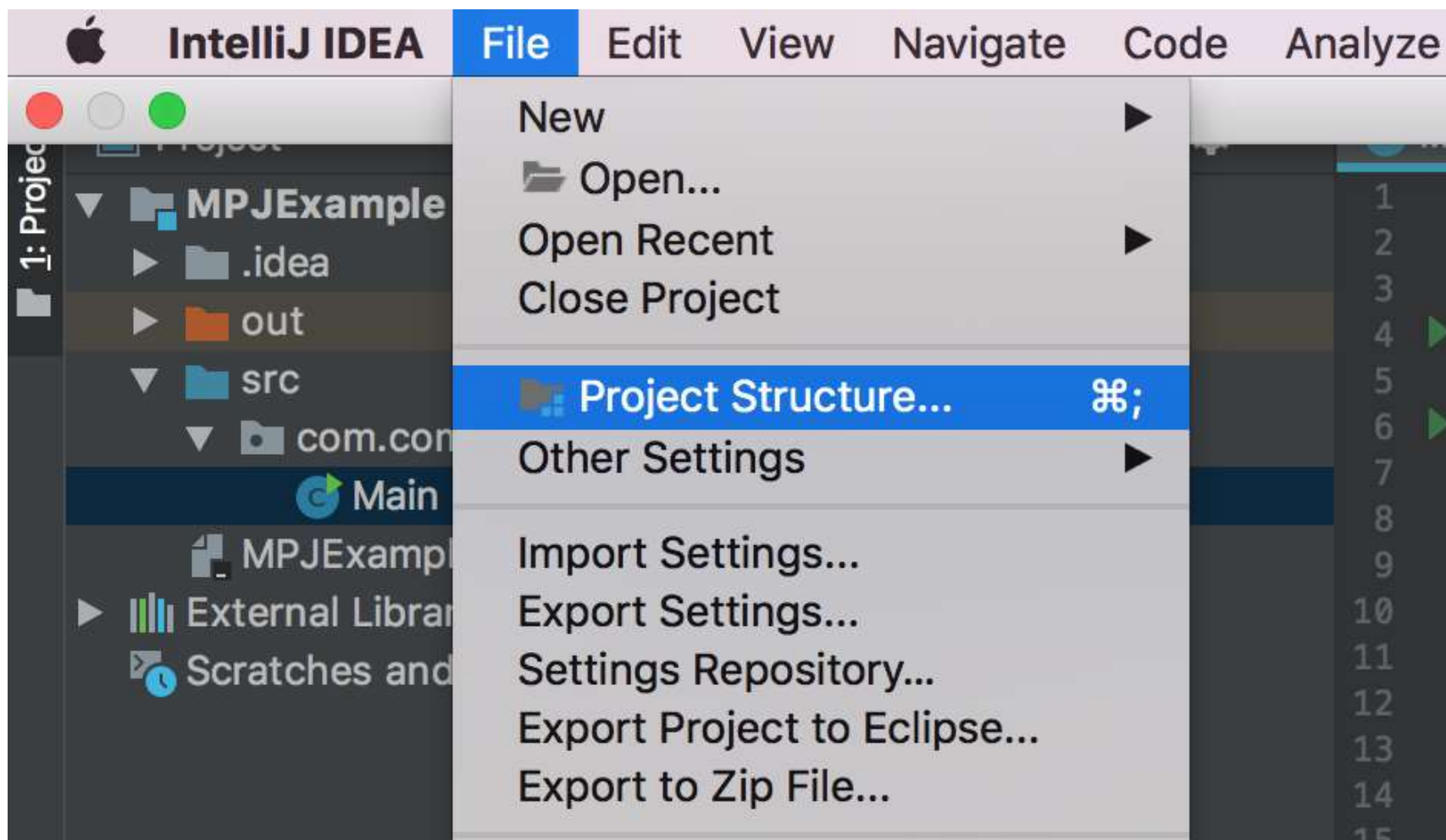
```
run:
MPJ Express (0.44) is started in the multicore configuration
Listening for transport dt_socket at address: 8000
Hi from <3>
Hi from <0>
Hi from <4>
```

A red arrow points from a yellow oval containing the text 'Отработали 3 из 5 процессов' to the 'Hi from <3>' line in the console output.

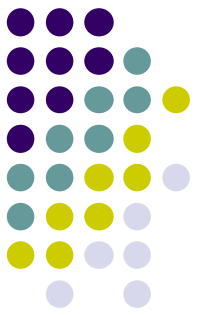
Настройка проекта в IntelliJ IDEA



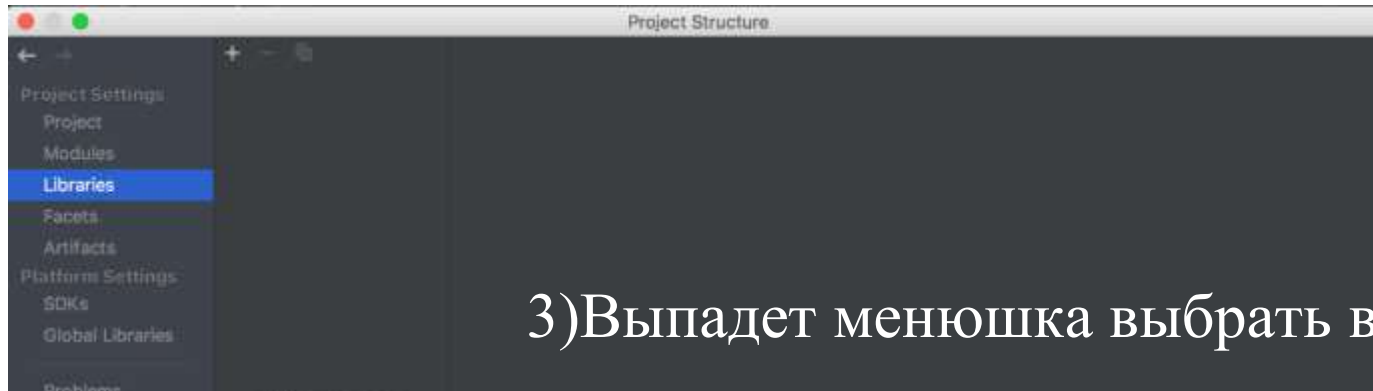
- 1)Выбрать File затем ProjectStructure:



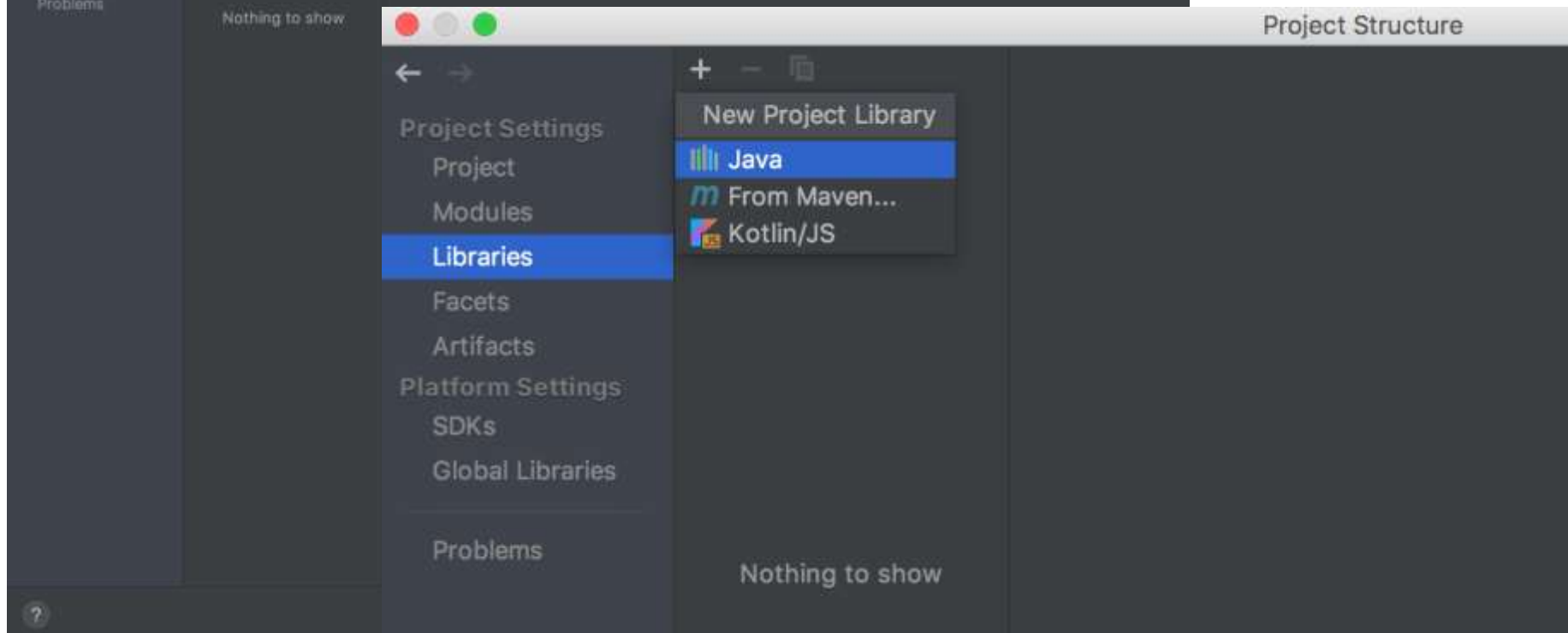
Настройка проекта в IntelliJ IDEA



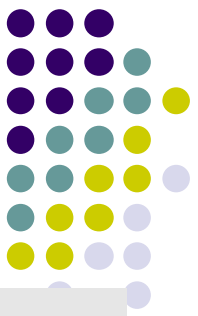
- 2) Выбрать Пункт Libraries и нажать на +:



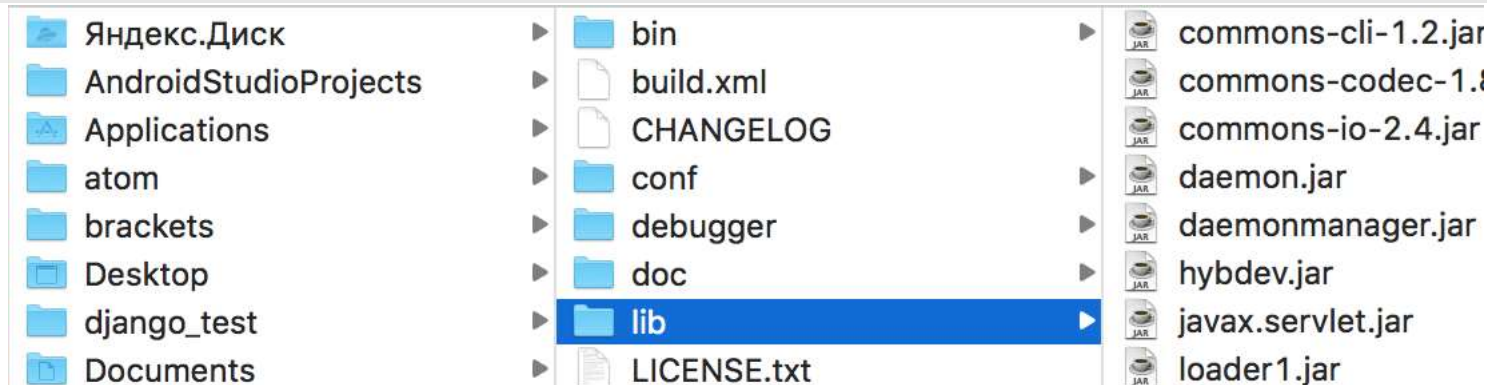
3) Выпадет менюшка выбрать в ней Java:



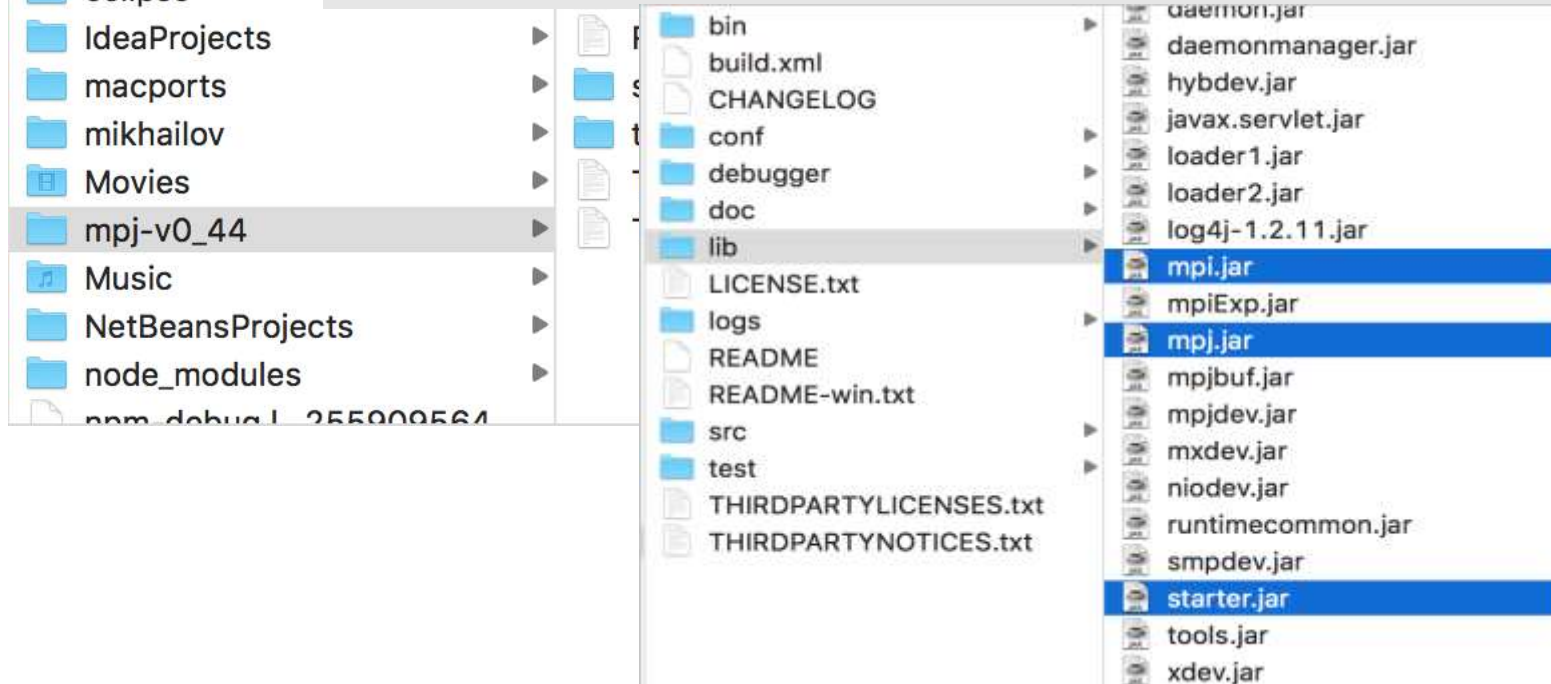
Настройка проекта в IntelliJ IDEA



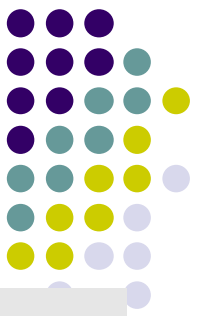
- 4) выбрать папку mpj в ней выбрать lib



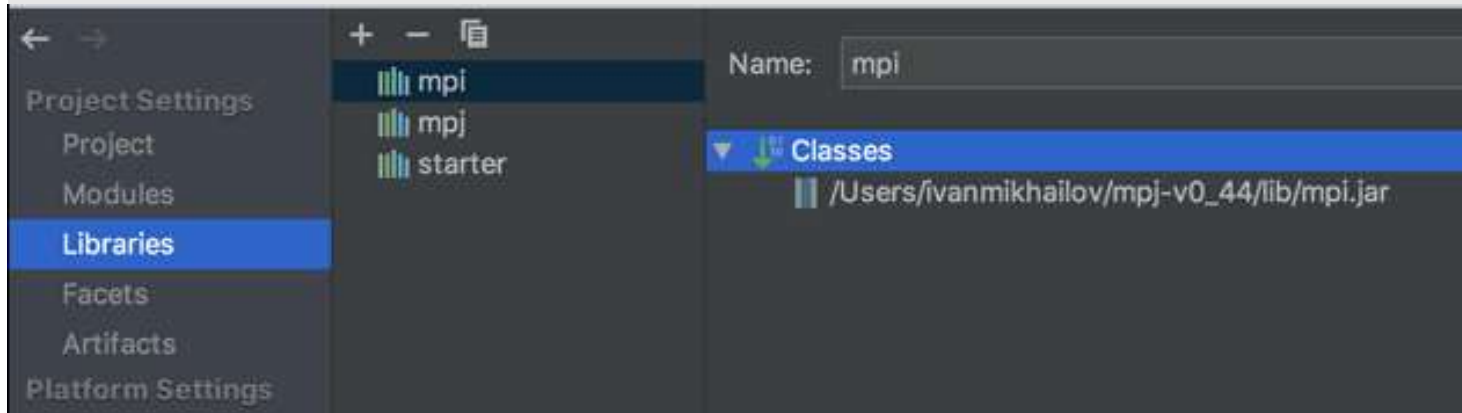
- 5) Выбрать файлы mpi.jar и starter.jar с расширением .jar



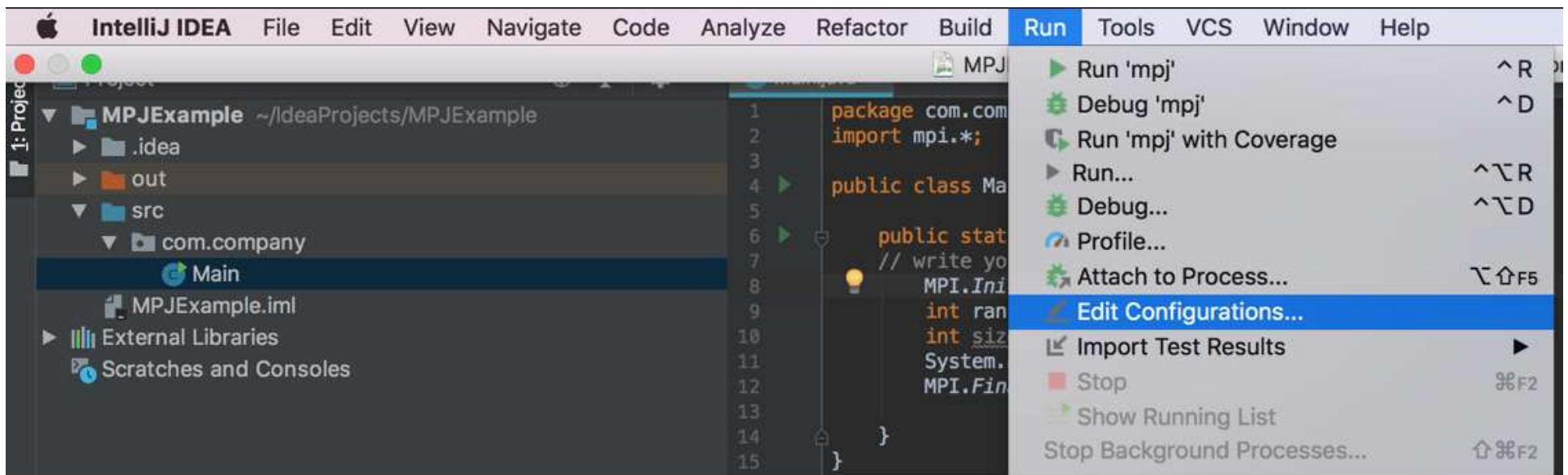
Настройка проекта в IntelliJ IDEA



- 6) В итоге должно быть так. Нажать apply и ок



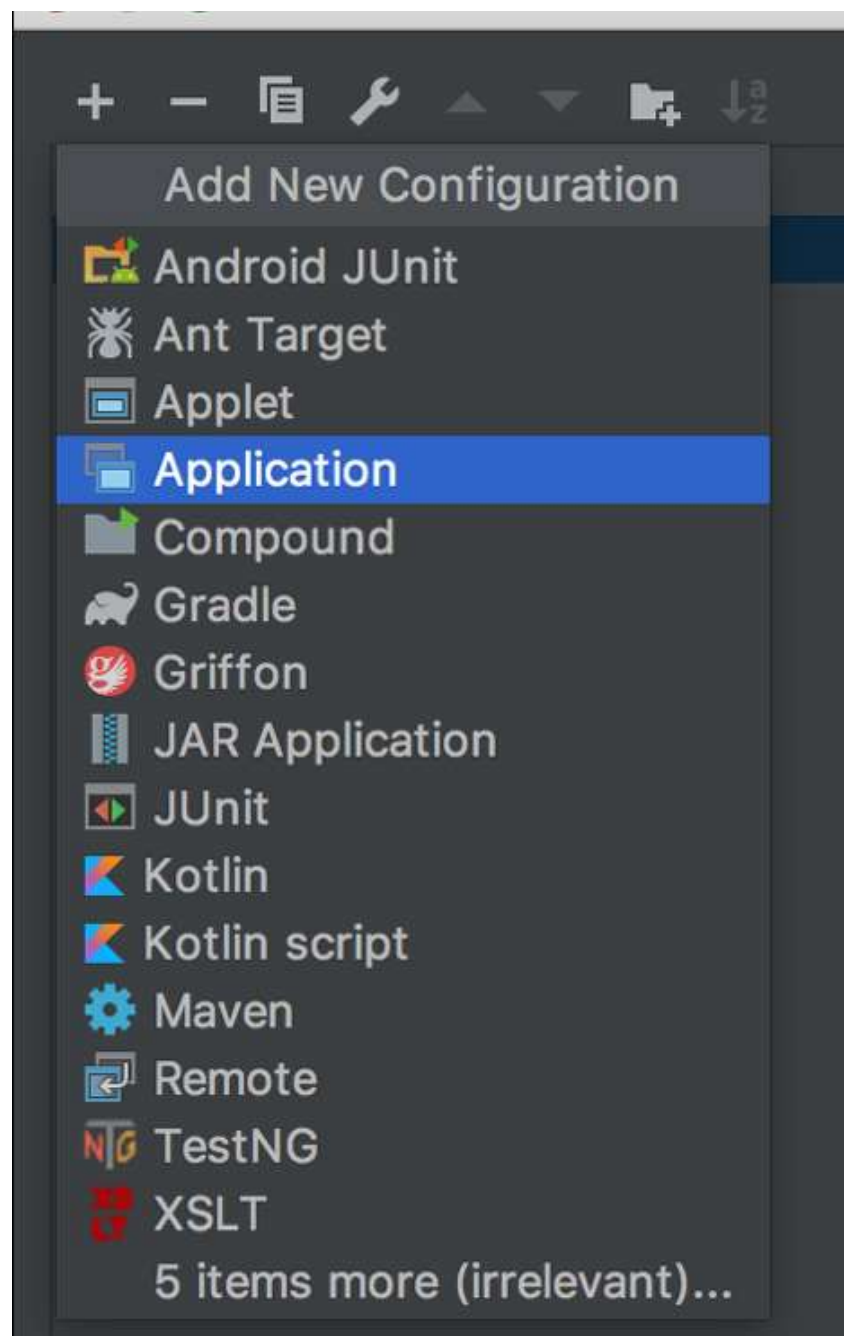
- 7) на вкладке run выбираем Edit Configurations, далее жмем +



Настройка проекта в IntelliJ IDEA



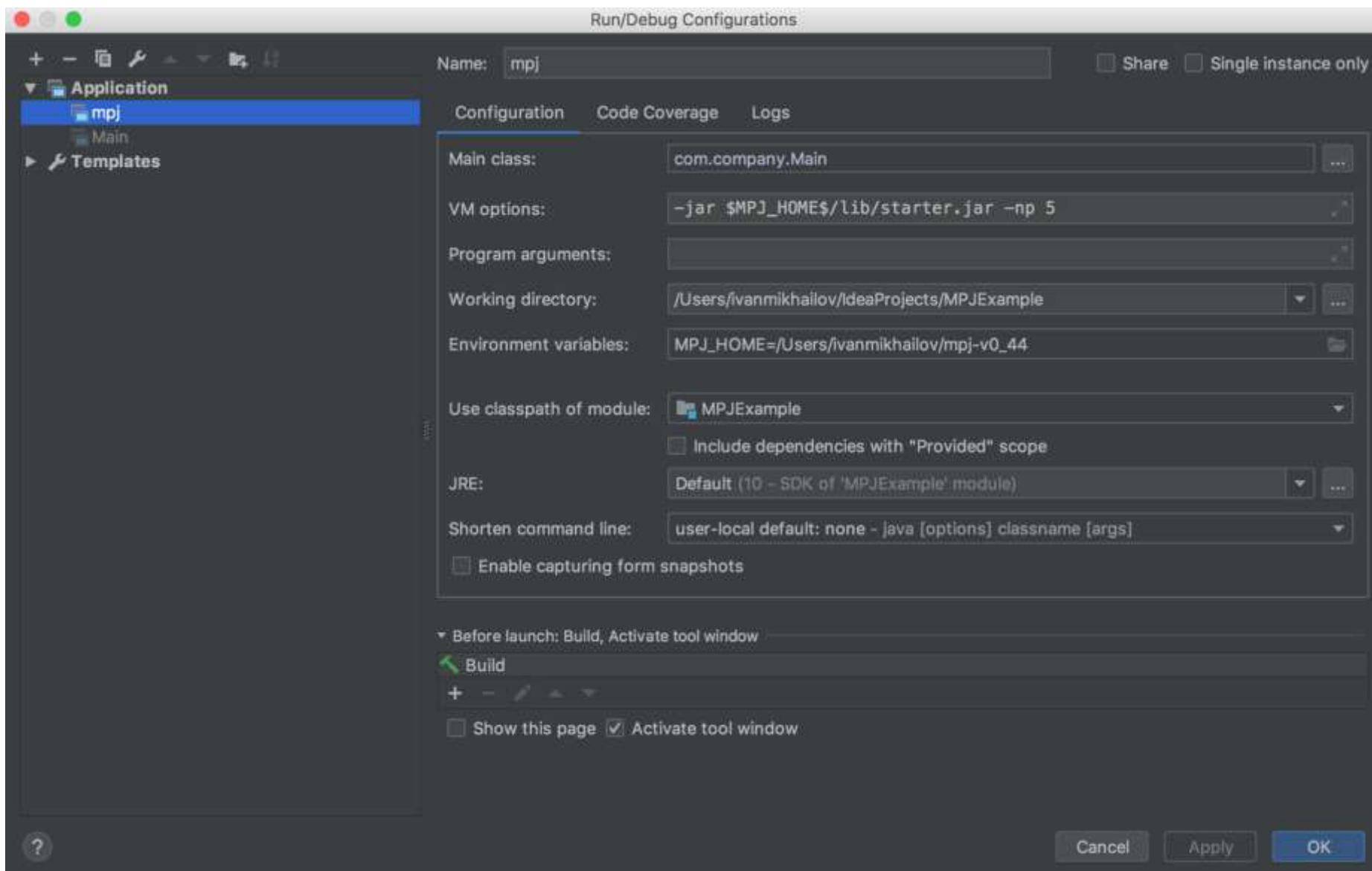
- 8) в открывшемся меню выбрать application



Настройка проекта в IntelliJ IDEA



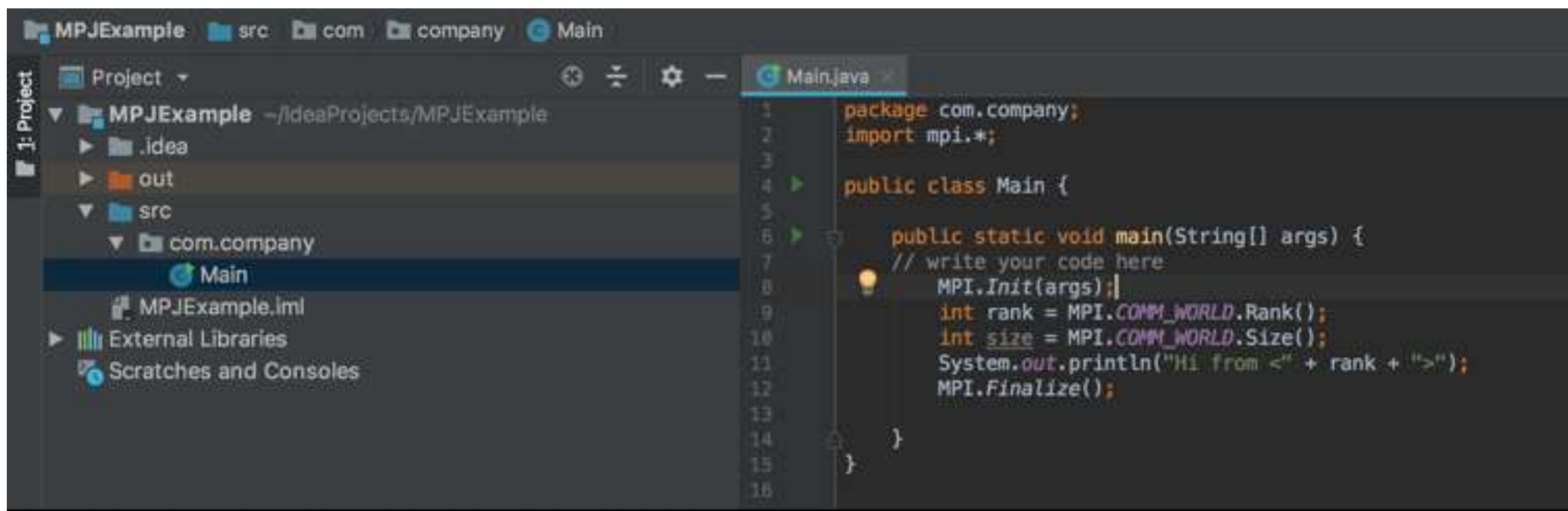
- 9)настроить_как_показано_на_экране_нажать_apply_и_ok



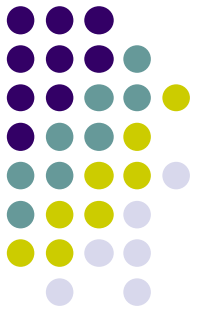
Настройка проекта в IntelliJ IDEA



- 10) Запустить проект и убедиться в его работоспособности



MPJ “Hello, World”



- **Можно скомпилировать из командной строки:**

```
javac -cp .:$MPJ_HOME/lib/mpj.jar MPJExample1.java
```

- **И запустить:**

```
mpjrun -np 4 MPJExample1
```

MPJ “Hello, World”

```
import mpi.*;
public class MPIAPP {

    public static void
    main(String[] args)
    throws Exception{

        MPI.Init(args);
        int rank =
        MPI.COMM_WORLD.Rank();

        int size =
        MPI.COMM_WORLD.Size();
        System.out.println("Hi
        from <"+rank+">");
        MPI.Finalize();
    }
}
```

MPI “Hello, World”

```
#include <stdio.h>
#include <mpi.h>
#include <iostream.h>

int main(int argc, char* argv[])
{
    int rank;size;
    MPI_Init(&argc, &argv);
    MPI_Comm_rank (MPI_COMM_WORLD, &rank);
    /* get current process id */

    MPI_Comm_size (MPI_COMM_WORLD, &size);
    /* get number of processes */
    printf( "Hello world from process %d of %d\n",
    rank, size );

    MPI_Finalize();
    system("pause");
    return 0;
}
```

Функции инициализации и завершения работы



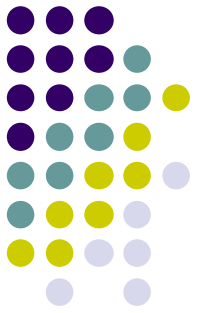
```
int MPI_Init(int* argc, char*** argv)
```

`argc` – указатель на счетчик аргументов командной строки

`argv` – указатель на список аргументов

```
int MPI_Finalize()
```

Структура программы на MPI



Структура *параллельной программы*, разработанная с использованием *MPI*, должна иметь следующий вид:

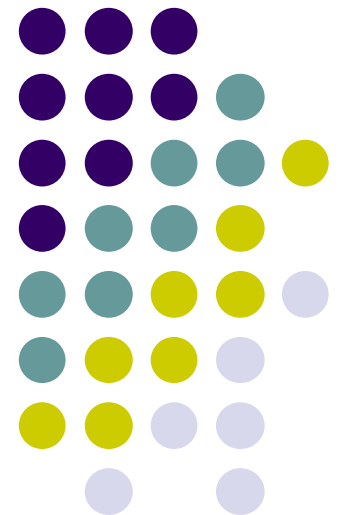
```
#include "mpi.h"
int main(int argc, char *argv[]) {
    <программный код без использования функций MPI>
    MPI_Init(&argc, &argv);
    <программный код с использованием функций MPI>
    MPI_Finalize();
    <программный код без использования функций MPI>
    return 0;
}
```

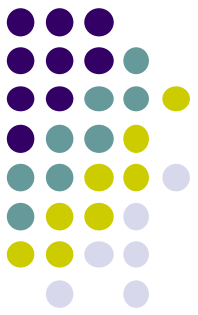
Полезные ссылки:



1. **MPJ Express: An Implementation of MPI in Java Windows User Guide 18th July 2014:**
mpj-express.org/docs/guides/windowsguide.pdf
2. **Debugging MPJ Express Applications using Eclipse and Netbeans in the multicore mode:**
mpjexpress.blogspot.ru/2010/12/debugging-parallel-applications-with.html
3. **QuickStart Guide: Running MPJ Express on UNIX/Linux/Mac platform Last Updated: Friday April 17 11:51:20 PKT 2015 Version 0.44** mpj-express.org/docs/readme/README
4. Документация по библиотеке MPJ:
<http://mpj-express.org/docs/javadocs/mapi/MPI.html>

Все задачи по курсу

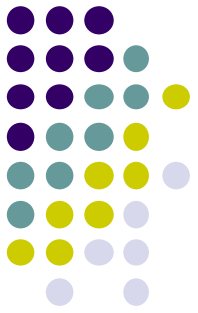




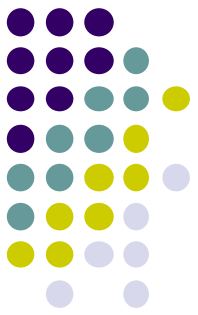
Задание 1

- В исходном тексте программы на языке С (след. слайд) предусмотрена некая схема обмена сообщениями между процессами параллельной программы.
- Определите схему обмена.
- В исходном тексте программы на языке С пропущены вызовы процедур двухточечного обмена. Предполагается, что при запуске четного числа процессов, те из них, которые имеют четный ранг, отправляют сообщение следующим по величине ранга процессам. Добавить эти вызовы, откомпилировать и запустить программу.

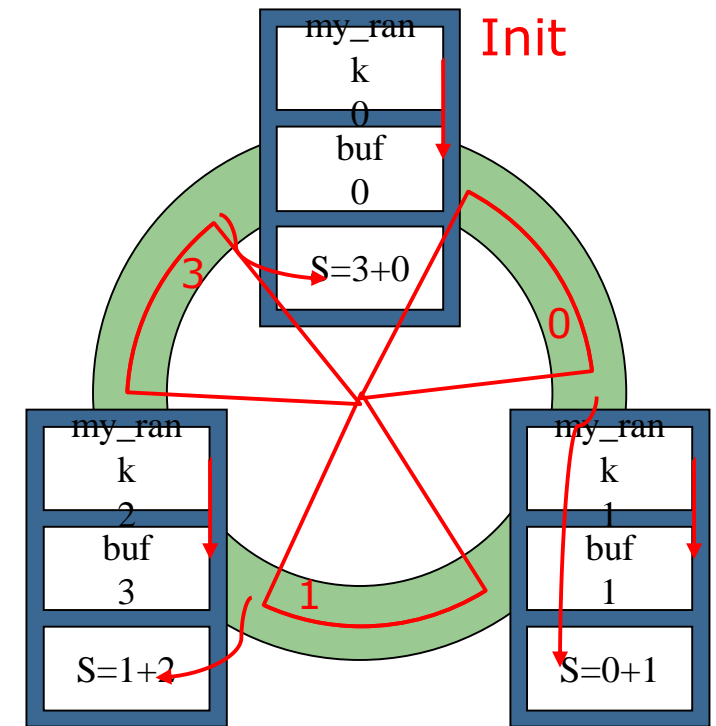
```
#include "mpi.h"
#include <stdio.h>
int main(int argc, char *argv[])
{
    int myrank, size, message;
    int TAG = 0;
    MPI_Status status;
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    message = myrank;
    if((myrank % 2) == 0)
    {
        if((myrank + 1) != size)
            MPI_Send(...);
    }
    else
    {
        if(myrank != 0)
            MPI_Recv(...);
        printf("received :%i\n", message);
    }
    MPI_Finalize();
    return 0;
}
```



Задание 2 — Пересылка данных по кольцу

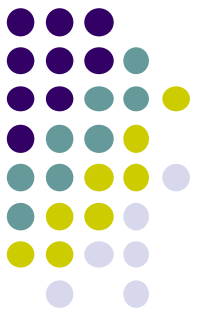


- Каждый процессор помещает свой ранг в целочисленную переменную *buf*.
- Каждый процессор пересылает переменную *buf* соседу справа (по часовой стрелке по кольцу).
- Каждый процессор суммирует принимаемое значение в переменную *s*, а затем передаёт рассчитанное значение соседу справа.
- Пересылки по кольцу прекращаются, когда нулевой процессор просуммирует ранги всех процессоров.



Выполнить 2 варианта: в блокирующем и неблокирующем режиме, там, где это возможно использовать `send_recieve`.

Неблокирующие обмены (пробники)

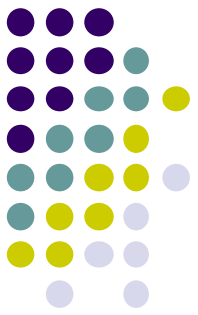


Задание 3.1

Дополнить программу с пробниками, выполнить ее.

```
int data[] = new int[1];
int buf[] = {1,3,5};
int count,TAG = 0;
Status st;
data[0] = 2016;
MPI.Init(arg);
int rank = MPI.COMM_WORLD.Rank();
int size=MPI.COMM_WORLD.Size();
if(rank == 0)
{
    MPI.COMM_WORLD.Send(data, 0, 1, MPI.INT, 2, TAG);
}
else if(rank == 1){
    MPI.COMM_WORLD.Send(buf, 0, buf.length, MPI.INT, 2, TAG);
}
```

Неблокирующие обмены (пробники)



Задание 3.1

(окончание)

```
else if(rank == 2){
    st = MPI.COMM_WORLD.Probe(...);
    count = st.Get_count(MPI.INT);
    MPI.COMM_WORLD.Recv(back_buf,0,count,MPI.INT,0,TAG);
    System.out.print("Rank = 0 ");
    for(int i = 0 ; i < count ; i ++){
        System.out.print(back_buf[i]+" ");
    }

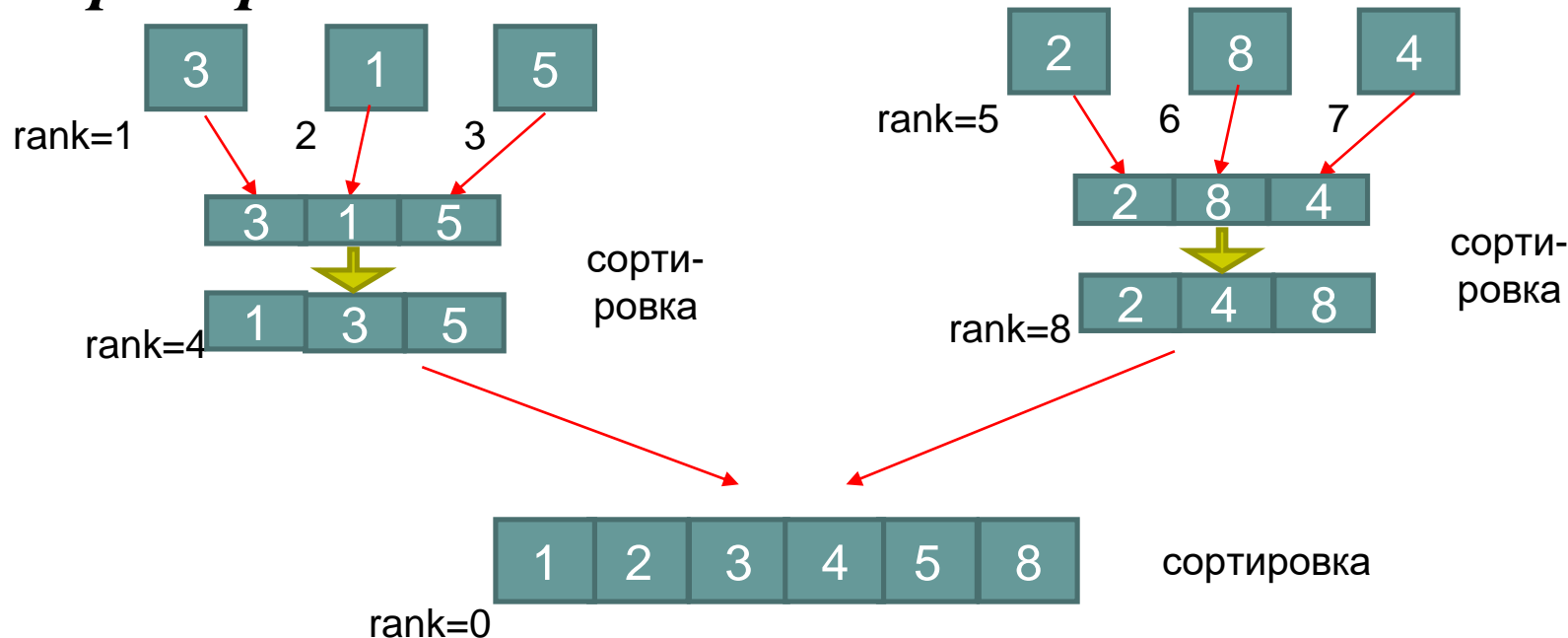
    st = MPI.COMM_WORLD.Probe(...);
    count = st.Get_count(MPI.INT);
    MPI.COMM_WORLD.Recv(back_buf2,0,count,MPI.INT,1,TAG);
    System.out.print("Rank = 1 ");
    for(int i = 0 ; i < count ; i ++){
        System.out.print(back_buf2[i]+" ");
    }
}
MPI.Finalize();
```

Неблокирующие обмены



Задание 3

Реализовать так называемую задачу фильтрации, используя неблокирующие обмены+Waitall()+пробники, например:



Вариант: к-во потоков 1-го уровня ($N_{\text{п}} \% 4$)+3

Более высокую оценку получают решения, предоставляющие возможность использовать переменное количество процессов.

Задание 4



Два вектора **a** и **b** размерности **N** представлены двумя одномерными массивами, содержащими каждый по **N** элементов. Напишите параллельную MPI-программу вычисления скалярного произведения этих векторов используя двухточечные обмены сообщениями (3 способа). Программа должна быть организована по схеме master-slave.

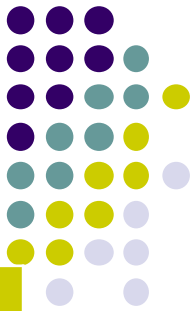
1. Измерьте и проанализируйте затраченное время на вычисления с кол-вом процессов 1-10 (количество элементов в векторах от 100, 1000, 10000).
2. Проведите исследование зависимости ускорения параллельной программы от числа процессоров (нарисовать графики).
3. Сделайте то же самое для других вариантов блокирующих обменов (синхронизированным, по готовности, и др. по желанию), постройте графики зависимости времени.
4. Проанализируйте вариант использования неблокирующих функций и реализуйте его.

Результаты сравнить, используя засечение времени с пом. функции `System.currentTimeMillis();` и оформить в виде графиков зависимости времени выполнения от числа процессов и ускорения от числа процессов. Результаты пояснить.

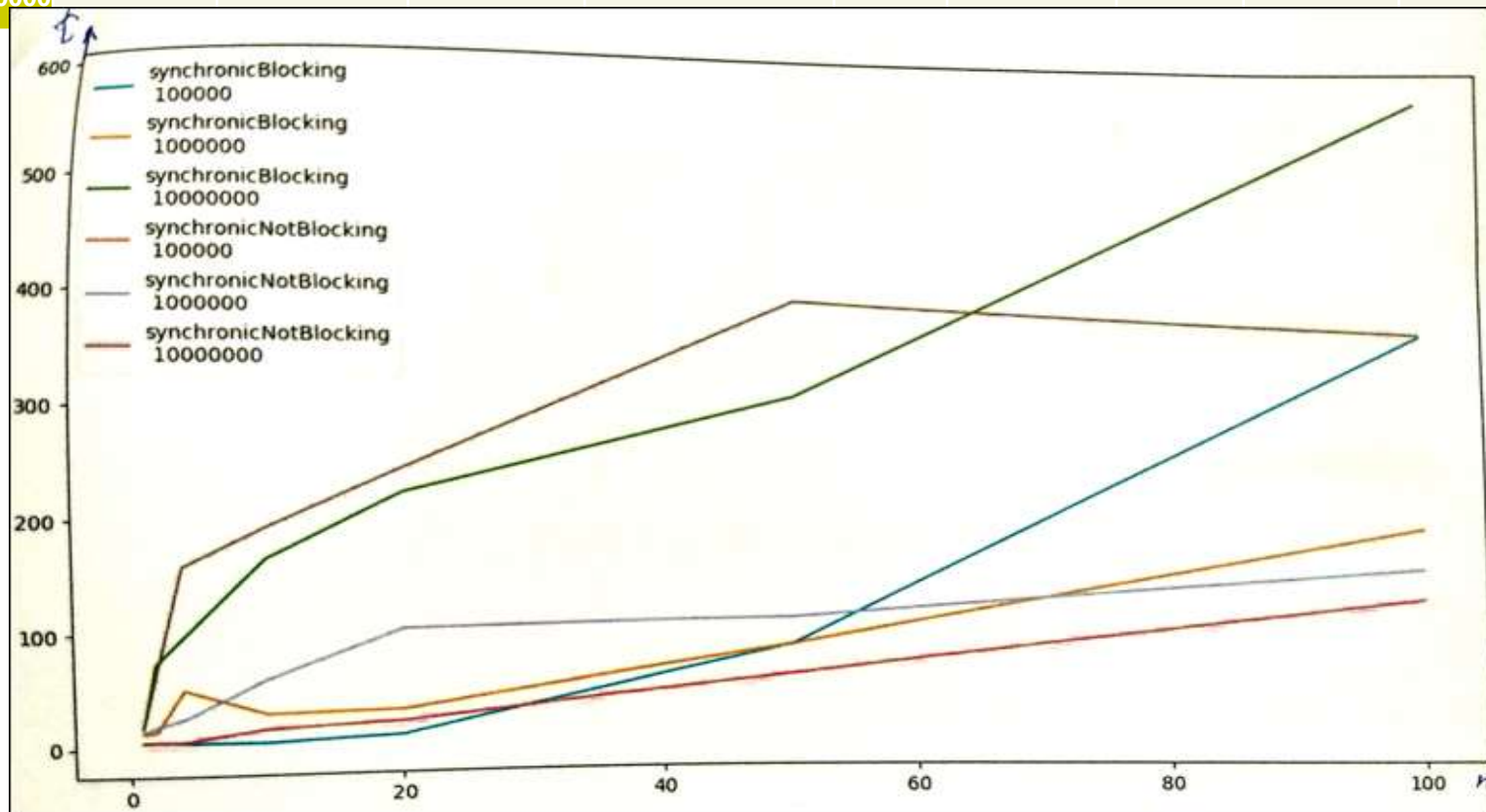
Выполняется по вариантам в таблице ниже. Задание сформулировано на основе векторов, каждому может достаться другая структура данных. Отчет с графиками обязателен!

Задание 4

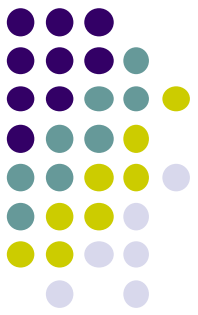
В качестве результата оформить таблицы и графики в виде:



Размерность Вектора/ матрицы	Последовательный алгоритм	Параллельный алгоритм, 2 процесса		4 процесса		10 процессов		50 процессов	
	Время выполнения	Время выполнения	Ускорение	Время выполнения	Ускор.	Время выполнения	Ускор.	Время выполнения	Ускор.
100									
1000									
10000									



Задание 4



Как измерить время.

Казалось бы, замерить время выполнения какого-то метода проще простого:

1. Вызываем один раз `System.currentTimeMillis` и сохраняем в переменную
2. Вызываем тестируемый метод и ждем его окончания
3. Вызываем еще раз `System.currentTimeMillis` и отнимаем результат из пункта 1

Но на самом деле такие математические операции со временем с помощью `System.currentTimeMillis` или других `java.time` методов вроде `Clock.now` может выдать некорректный результат.

Почему?

Все потому, что время от времени срабатывает механизм обновления машинных часов, где работает ваше Java приложение. И это обновление может запуститься в любой момент. А по закону подлости это случится как раз тогда, когда вы будете вызывать тестируемый метод из пункта 2, что приведет к ошибкам в расчетах.

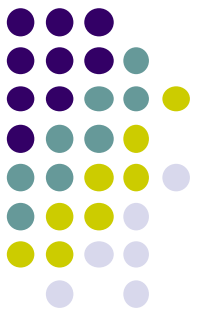
Как тогда замерять?

Есть несколько вариантов, но наиболее предпочтительным для меня является класс `Stopwatch` из пакета `com.google.common.base`.

На картинке продемонстрировано, как его просто и удобно использовать.

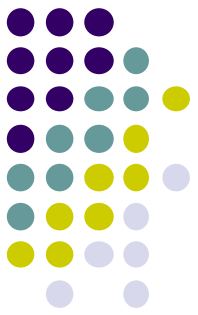
Задание 4

Как измерить время.



```
Stopwatch stopwatch = Stopwatch.createStarted();  
try {  
    eventProcessor.process(new Event());  
} finally {  
    Duration duration = stopwatch.elapsed();  
    System.out.println("Time: " + duration);  
}
```


Задание 4



Два вектора a и b размерности N представлены двумя одномерными массивами, содержащими каждый по N элементов. Решите задачу о нахождении скалярного произведения векторов A и B с учетом знания принципов коллективных обменов

- с помощью функции Broadcast/Reduse;
- с помощью функций Scatter(v) / Gather(v).

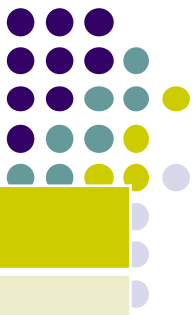
Результаты сравнить, используя засечение времени с пом. функции `System.currentTimeMillis()`; и оформить в виде графиков зависимости времени выполнения от числа процессов и ускорения от числа процессов. Результаты пояснить.

Выполняется по вариантам изложенным в таблице ниже. Задание сформулировано на основе векторов, Вам может достаться другая структура данных.

Отчет с графиками обязателен!

Задание 4

Варианты



№	Задание
1	Разработать алгоритм вычисления произведения матрицы на вектор. Учесть наличие хвоста.
2	Разработать алгоритм вычисления произведения матриц (схема А)
3	Разработать алгоритм вычисления произведения матриц (схема В)
4	Вычисление скалярного произведения векторов, a – рассылается всем равными частями, b – передается по кольцу. Учесть наличие хвоста.
5	Разработать алгоритм умножения матрицы на вектор (схема А)
6	Разработать алгоритм умножения матрицы на вектор (схема В)
7	Вычисление скалярного произведения векторов, a и b – рассылаются всем процессам равными частями, . Учесть наличие хвоста.

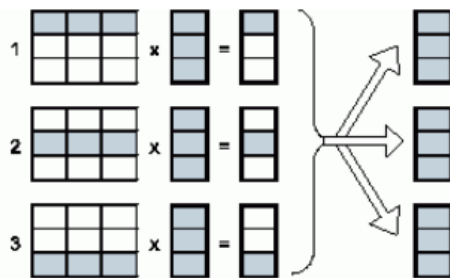
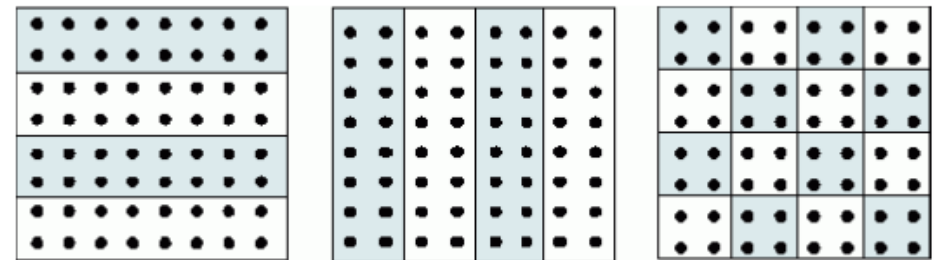


Схема вычисления произведения матрицы на вектор (для схемы А)



- А) Ленточная горизонталь для матрицы А
- В) Ленточная вертикаль для матрицы A_{67}
- С) Блочное деление

Задание 4

Описание алгоритма 1 – схема А

Вычислить произведение матриц $C=A \times B$, где A – матрица размера $n_1 \times n_2$ и B – матрица $n_2 \times n_3$. Матрица результатов C имеет размер $n_1 \times n_3$. Исходные матрицы первоначально доступны на нулевом процессе, и матрица результатов возвращается в нулевой процесс.

Матрицы разрезаны, как показано на рис. 1: матрица A разрезана на p_1 горизонтальных полос, матрица B разрезана на p_2 вертикальных полос, и матрица результата C разрезана на $p_1 \times p_2$ подматриц (или субматрицы).

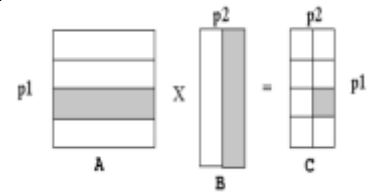


рис. 1.

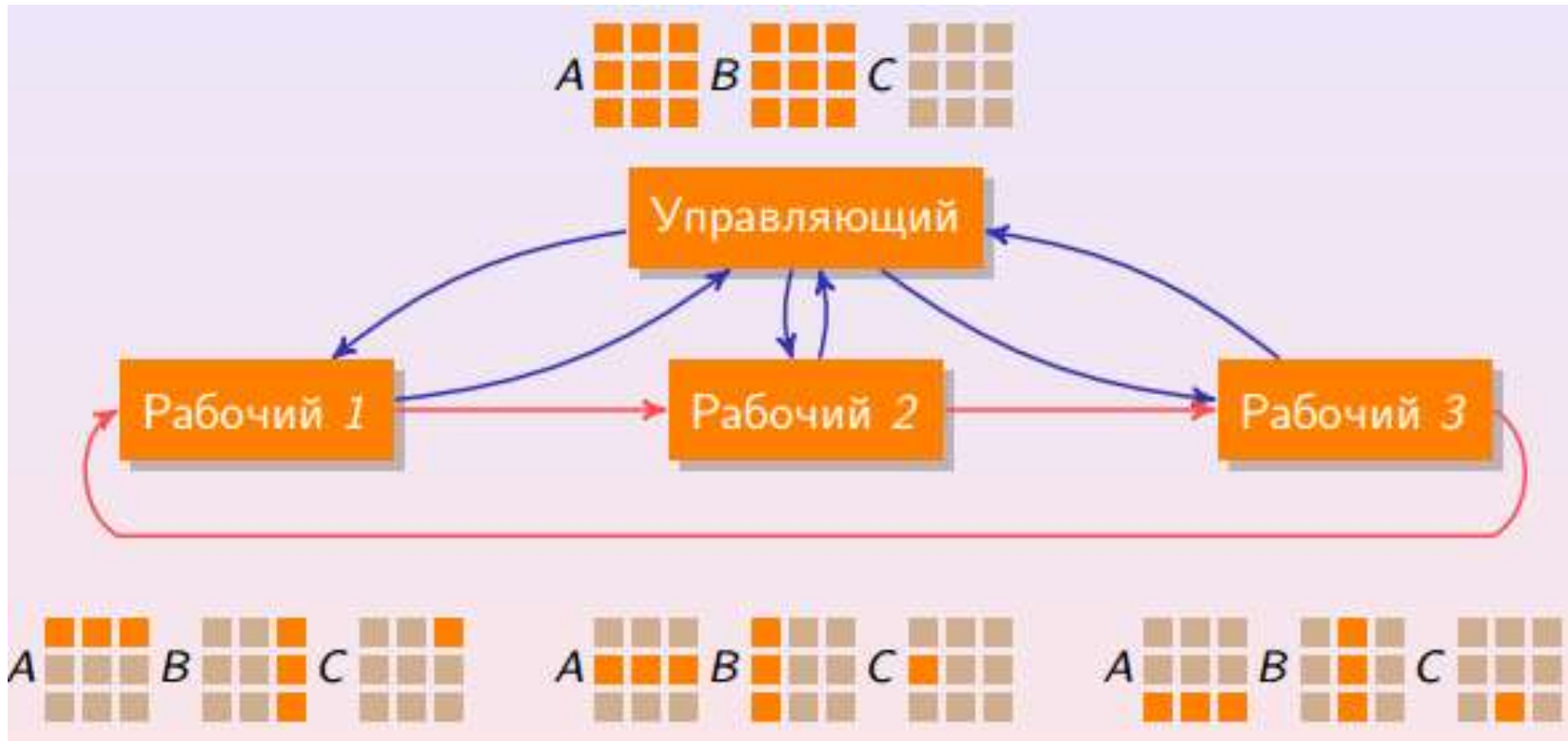
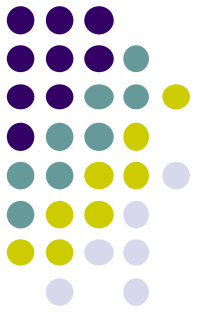
Каждый процесс (i,j) вычисляет произведение i -й горизонтальной полосы матрицы A и j -й вертикальной полосы матрицы B , произведение получено в подматрице (i,j) матрицы C . Если число процессов (i,j) не кратно числу строк матрицы ($i < n_1$, $j < n_2$), то процессу пересылается несколько строк матрицы A и он формирует несколько ячеек матрицы C . Фактически каждая подзадача сводится к скалярному умножению векторов.

Для выполнения всех необходимых вычислений базовой подзадаче должны быть доступны одна из строк матрицы A и все столбцы матрицы B . Простое решение этой проблемы – дублирование матрицы B во всех подзадачах – является, как правило, неприемлемым в силу больших затрат памяти для хранения данных. Поэтому организация вычислений должна быть построена таким образом, чтобы в каждый текущий момент времени подзадачи содержали лишь часть данных, необходимых для проведения расчетов, а доступ к остальной части данных обеспечивался бы при помощи передачи данных между процессорами.

Задание 4

Описание алгоритма 1 – схема А

Модель управляющий-рабочий 1



Подзадачи осуществляют обмен столбцами, в ходе которого каждая подзадача передает свой столбец матрицы В следующей подзадаче в соответствии с кольцевой структурой информационных взаимодействий.

Задание 4

Описание алгоритма 2 – схема В

Вычислить произведение матриц $C=A \times B$, где A – матрица размера $n_1 \times n_2$ и B – матрица $n_2 \times n_3$. Матрица результатов C имеет размер $n_1 \times n_3$. Исходные матрицы первоначально доступны на нулевом процессе, и матрица результатов возвращается в нулевой процесс.

Матрицы разрезаны, как показано на рис. 2: матрица A разрезана на p_1 горизонтальных полос, матрица B разрезана на p_2 горизонтальных полос, и матрица результата C разрезана на $p_1 \times p_2$ подматриц.

Отличие второго алгоритма (В) состоит в том, что в подзадачах располагаются не столбцы, а строки матрицы B . Как результат, перемножение данных каждой подзадачи сводится не к скалярному умножению имеющихся векторов, а к их поэлементному умножению. В результате подобного умножения в каждой подзадаче получается строка частичных результатов для матрицы C .

При рассмотренном способе разделения данных для выполнения операции матричного умножения нужно обеспечить последовательное получение в подзадачах всех строк матрицы B , поэлементное умножение данных и суммирование вновь получаемых значений с ранее вычисленными результатами. Организация необходимой последовательности передач строк матрицы B между подзадачами также может быть выполнена с использованием кольцевой структуры информационных связей.

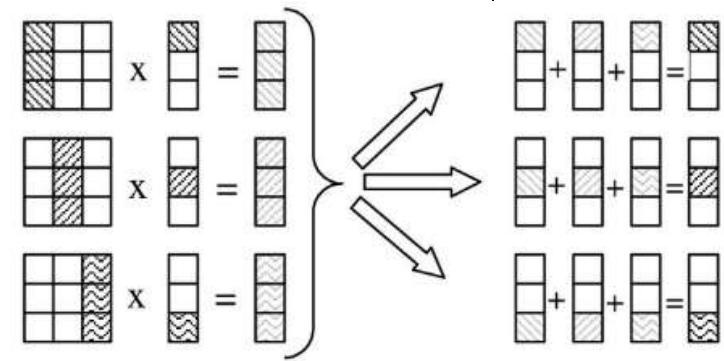
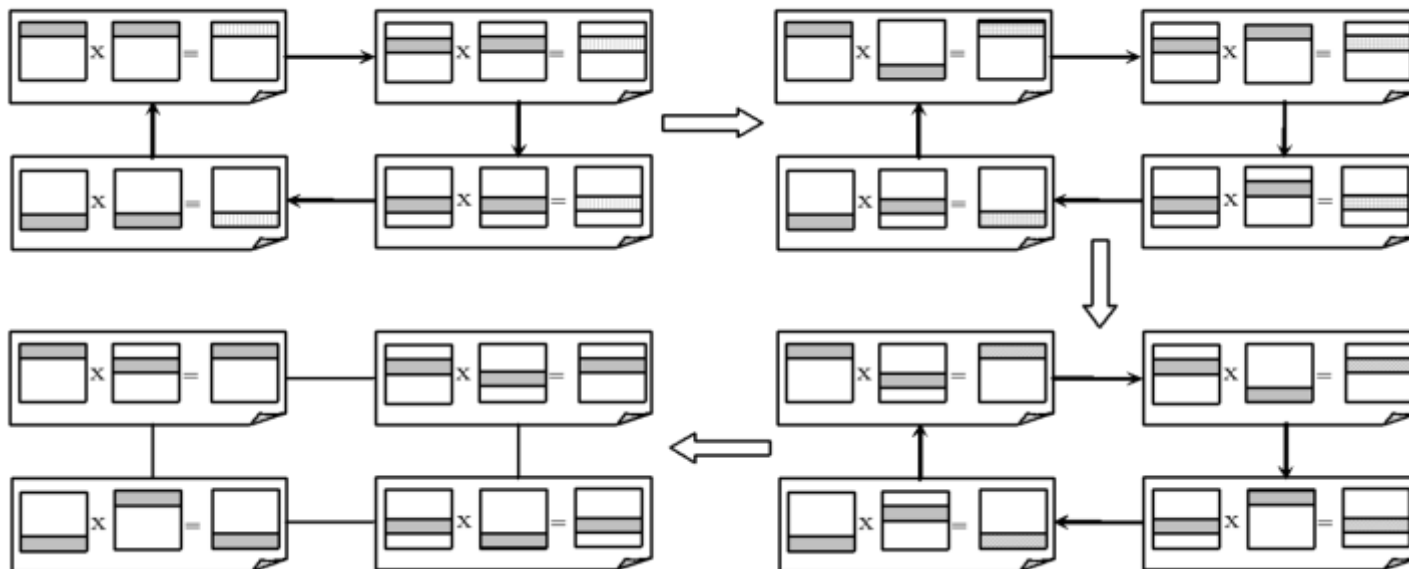


рис. 2. – базовая подзадача

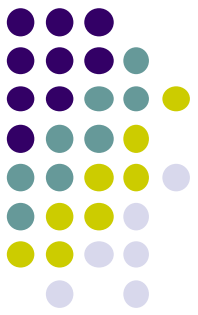
Задание 4

Описание алгоритма 1 – схема В

Параллельный алгоритм умножения матрицы на вектор начинается с того, что каждая базовая задача i выполняет умножение своего столбца матрицы A на элемент b_i , в итоге в каждой подзадаче получается вектор $s'(i)$ промежуточных результатов. Далее для получения элементов результирующего вектора C подзадачи должны обмениваться своими промежуточными данными между собой (элемент j , $0 \leq j < n$, частичного результата $s'(i)$ подзадачи i , $0 \leq i < n$, должен быть передан подзадаче j). Данная обобщенная передача данных (all-to-all communication или total exchange) является наиболее общей коммуникационной процедурой и может быть реализована при помощи функции `MPI_Alltoall` библиотеки `MPI`. После выполнения передачи данных каждая базовая подзадача i , $0 \leq i < n$, будет содержать n частичных значений $s'(i(j))$, $0 \leq j < n$, сложением которых и определяется элемент s_i вектора результата C .



Задание 5



5. Задачи с графами.

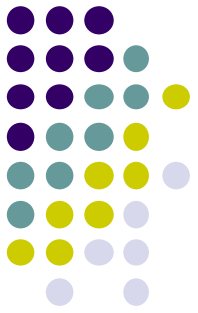
Задание выполняется по вариантам: Вариант определяет преподаватель. Каждый студент должен выполнить по 1 задаче из таблицы. Входные данные – матрица смежности.

Исходные данные сформировать так, чтоб их легко было проверить (возможно Вам понадобятся несколько вариантов исходных данных).

Выполнить подсчет временных затрат с разным кол-вом процессов, привести графики. ОТЧЕТ ОБЯЗАТЕЛЕН.

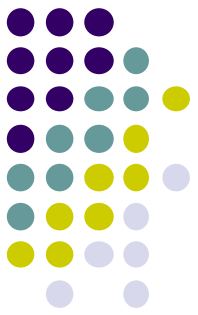
№	Задание
0	Разработать алгоритм вычисления диаметра произвольного неориентированного графа.
1	Разработать алгоритм вычисления максимальной из степеней вершин в графе.
2	Разработать алгоритм вычисления количества ребер в графе.
3	Разработать алгоритм вычисления центра графа
4	Разработать алгоритм определения того, является ли граф деревом.
5	Разработать алгоритм определения того, является ли граф тором.
6	Разработать алгоритм определения того, является ли граф гиперкубом.
7	Разработать алгоритм определения того, является ли граф регулярным.

Задание 6



- Смоделировать работу блокчейна.
- Задачу можно выполнять в паре.

Задание 7



- Разработать Spring-boot распределенное приложение:
- **Необходимо самостоятельно выполнить:**
- Создание проекта, содержащего реализацию классов на языке Java, описывающих предметную область согласно варианту.
- Обеспечение возможности работы с множеством объектов описанных классов в виде списка или динамического массива, запись данных объектов в файл формата согласно варианту и чтение из него.
- Разработку методов http-запросов для управления объектами: добавление, удаление, вывод объекта (например, по ID), вывод всех объектов, сохранение данных в файл, загрузка данных из файла, вывод сведений о приложении, включая расчёт, статистических величин для имеющегося множества объектов.
- Разработанное приложение не должно допускать возникновение необрабатываемых исключений и непреднамеренного завершения работы.