Zarzadzanie_Zespolem_Naukowym

Generated by Doxygen 1.14.0

Chapter 1

Namespace Index

1.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

Chapter 5

Namespace Documentation

5.1 enums Namespace Reference

Enumerations

- enum class Sex { Male , Female }

  Enumeration for sex.
- enum class Rank { Trainee , Junior , Senior }

  Enumeration for academic rank.
- enum class FieldofResearchName {
  Physics , Chemistry , Biology , ComputerScience ,
  Mathematics , Medicine , Psychology , SocialSciences ,
  Economics , Astronomy , Neuroscience , Robotics ,
  Geology , Anthropology }

  Enumeration for field of research names.

5.1.1 Enumeration Type Documentation

5.1.1.1 FieldofResearchName

```
enum class enums::FieldofResearchName  [strong]
```

Enumeration for field of research names.

Enumerator

Physics	Physics.
Chemistry	Chemistry.
Biology	Biology.
ComputerScience	Computer Science.
Mathematics	Mathematics.
Medicine	Medicine.
Psychology	Psychology.
SocialSciences	Social Sciences.
Economics	Economics.

Astronomy	Astronomy.
Neuroscience	Neuroscience.
Robotics	Robotics.
Geology	Geology.
Anthropology	Anthropology.

Definition at line 21 of file enums.h.

5.1.1.2 Rank

```
enum class enums::Rank  [strong]
```

Enumeration for academic rank.

Enumerator

Trainee	Trainee rank.
Junior	Junior rank.
Senior	Senior rank.

Definition at line 14 of file enums.h.

5.1.1.3 Sex

```
enum class enums::Sex  [strong]
```

Enumeration for sex.

Enumerator

Male	Male sex.
Female	Female sex.

Definition at line 8 of file enums.h.

Chapter 6

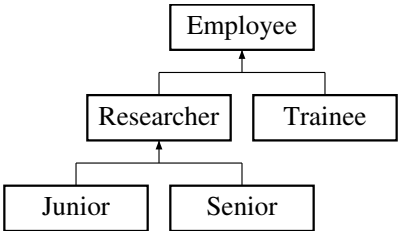
Class Documentation

6.1 Employee Class Reference

Abstract base class representing an employee.

`#include <employee.h>`

Inheritance diagram for Employee:



Public Member Functions

- Employee (const std::string &name, const std::string &surname, int age, enums::Sex sex, enums::Rank rank, const FieldofResearch &field)

  Constructs an Employee object.
- virtual void show () const =0

  Displays employee information (pure virtual).
- virtual double getSalary () const =0

  Returns the employee's salary (pure virtual).
- virtual ~Employee ()=default

  Virtual destructor.
- const std::string & getName () const

  Returns the employee's first name.
- const std::string & getSurname () const

  Returns the employee's surname.
- int getAge () const

  Returns the employee's age.
- enums::Sex getSex () const

  Returns the employee's sex.

- enums::Rank getRank () const

  Returns the employee's rank.
- const FieldofResearch & getFieldofresearch () const

  Returns the employee's field of research.
- const std::string sexToString () const

  Returns the employee's sex as a string.
- std::string getFieldNameAsString () const

  Returns the employee's field name as a string.
- std::string getDescriptionAsString () const

  Returns the employee's field description as a string.

6.1.1 Detailed Description

Abstract base class representing an employee.

This class defines the common interface and properties for all employees.

Definition at line 38 of file employee.h.

6.1.2 Constructor & Destructor Documentation

6.1.2.1 Employee()

```
Employee::Employee (
            const std::string & name,
            const std::string & surname,
            int age,
            enums::Sex sex,
            enums::Rank rank,
            const FieldofResearch & field)
```

Constructs an Employee object.

Parameters

name	Employee's first name.
surname	Employee's surname.
age	Employee's age.
sex	Employee's sex.
rank	Employee's rank.
field	Employee's field of research.

Definition at line 4 of file employee.cpp.

6.1.2.2 ~Employee()

```
virtual Employee::~Employee ()  [virtual], [default]
```

Virtual destructor.

6.1.3 Member Function Documentation

6.1.3.1 getAge()

```
int Employee::getAge () const
```

Returns the employee's age.

Definition at line 110 of file employee.cpp.

6.1.3.2 getDescriptionAsString()

```
std::string Employee::getDescriptionAsString () const
```

Returns the employee's field description as a string.

Definition at line 100 of file employee.cpp.

6.1.3.3 getFieldNameAsString()

```
std::string Employee::getFieldNameAsString () const
```

Returns the employee's field name as a string.

Definition at line 97 of file employee.cpp.

6.1.3.4 getFieldofresearch()

```
const FieldofResearch & Employee::getFieldofresearch () const
```

Returns the employee's field of research.

Definition at line 119 of file employee.cpp.

6.1.3.5 getName()

```
const std::string & Employee::getName () const
```

Returns the employee's first name.

Definition at line 104 of file employee.cpp.

6.1.3.6 getRank()

```
enums::Rank Employee::getRank () const
```

Returns the employee's rank.

Definition at line 116 of file employee.cpp.

6.1.3.7 getSalary()

```
virtual double Employee::getSalary () const  [pure virtual]
```

Returns the employee's salary (pure virtual).

Implemented in Junior, Senior, and Trainee.

6.1.3.8 getSex()

```
enums::Sex Employee::getSex () const
```

Returns the employee's sex.

Definition at line 113 of file employee.cpp.

6.1.3.9 getSurname()

```
const std::string & Employee::getSurname () const
```

Returns the employee's surname.

Definition at line 107 of file employee.cpp.

6.1.3.10 sexToString()

```
const std::string Employee::sexToString () const
```

Returns the employee's sex as a string.

Definition at line 86 of file employee.cpp.

6.1.3.11 show()

```
virtual void Employee::show () const  [pure virtual]
```

Displays employee information (pure virtual).

Implemented in Junior, Senior, and Trainee.

The documentation for this class was generated from the following files:

- employee.h
- employee.cpp

6.2 FieldofResearch Class Reference

Represents a scientific field of research.

```
#include <employee.h>
```

Public Member Functions

- FieldofResearch (enums::FieldofResearchName fieldName, double points)

  Constructs a FieldofResearch object.
- std::string getDescription () const

  Returns the description of the research field.
- const enums::FieldofResearchName getFieldName () const

  Returns the name of the research field as an enum.
- const double getPoints () const

  Returns the points associated with the research field.
- std::string getFieldNametoString () const

  Returns the name of the research field as a string.

6.2.1 Detailed Description

Represents a scientific field of research.

This class stores information about the research field, including its name, description, and associated points.

Definition at line 10 of file employee.h.

6.2.2 Constructor & Destructor Documentation

6.2.2.1 FieldofResearch()

```
FieldofResearch::FieldofResearch (
            enums::FieldofResearchName fieldName,
            double points)
```

Constructs a FieldofResearch object.

Parameters

fieldName	Name of the research field.
points	Points associated with the field.

Definition at line 7 of file employee.cpp.

6.2.3 Member Function Documentation

6.2.3.1 getDescription()

```
std::string FieldofResearch::getDescription () const
```

Returns the description of the research field.

Definition at line 11 of file employee.cpp.

6.2.3.2 getFieldName()

```
const enums::FieldofResearchName FieldofResearch::getFieldName () const
```

Returns the name of the research field as an enum.

Definition at line 45 of file employee.cpp.

6.2.3.3 getFieldNametoString()

```
std::string FieldofResearch::getFieldNametoString () const
```

Returns the name of the research field as a string.

Definition at line 51 of file employee.cpp.

6.2.3.4 getPoints()

```
const double FieldofResearch::getPoints () const
```

Returns the points associated with the research field.

Definition at line 48 of file employee.cpp.

The documentation for this class was generated from the following files:

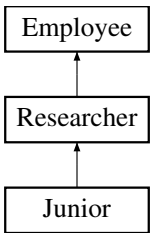
- employee.h
- employee.cpp

6.3 Junior Class Reference

Represents a junior researcher.

```
#include <rank.h>
```

Inheritance diagram for Junior:



Public Member Functions

- Junior (const std::string &name, const std::string &surname, int age, enums::Sex sex, enums::Rank rank, const FieldofResearch &field, const std::vector< Publication > &pub, int ministralPoints)

  Constructs a Junior object.
- void show () const override

  Displays junior researcher information.
- double getSalary () const override

  Returns the junior researcher's salary.

Public Member Functions inherited from Researcher

- void getPublicationsAndCitations () const

  Displays the researcher's publications and citations.
- size_t publicationCount () const

  Returns the number of publications.
- int getHirshIndex () const

  Calculates and returns the Hirsch index (h-index).
- int getMinistralPoints () const

  Returns the ministral points.
- const std::vector< Publication > & getPublications () const

  Returns a const reference to the list of publications.
- Researcher (const std::string &name, const std::string &surrname, int age, enums::Sex sex, enums::Rank rank, const FieldofResearch &field, const std::vector< Publication > &pub, int ministralPoints)

  Constructs a Researcher object.

Public Member Functions inherited from Employee

- Employee (const std::string &name, const std::string &surname, int age, enums::Sex sex, enums::Rank rank, const FieldofResearch &field)

  Constructs an Employee object.
- virtual ~Employee ()=default

  Virtual destructor.
- const std::string & getName () const

  Returns the employee's first name.
- const std::string & getSurname () const

  Returns the employee's surname.
- int getAge () const

  Returns the employee's age.
- enums::Sex getSex () const

  Returns the employee's sex.
- enums::Rank getRank () const

  Returns the employee's rank.
- const FieldofResearch & getFieldofresearch () const

  Returns the employee's field of research.
- const std::string sexToString () const

  Returns the employee's sex as a string.
- std::string getFieldNameAsString () const

  Returns the employee's field name as a string.
- std::string getDescriptionAsString () const

  Returns the employee's field description as a string.

Additional Inherited Members

Protected Attributes inherited from Researcher

- int ministralPoints

  Ministral points of the researcher.
- std::vector< Publication > publicationsAndCitations

  List of publications and their citation counts.

6.3.1 Detailed Description

Represents a junior researcher.

Inherits from Researcher.

Definition at line 45 of file rank.h.

6.3.2 Constructor & Destructor Documentation

6.3.2.1 Junior()

```
Junior::Junior (
            const std::string & name,
            const std::string & surname,
            int age,
            enums::Sex sex,
            enums::Rank rank,
            const FieldofResearch & field,
            const std::vector< Publication > & pub,
            int ministralPoints)
```

Constructs a Junior object.

Parameters

name	First name.
surname	Surname.
age	Age.
sex	Sex.
rank	Rank.
field	Field of research.
pub	List of publications.
ministralPoints	Ministral points.

Definition at line 39 of file rank.cpp.

6.3.3 Member Function Documentation

6.3.3.1 getSalary()

```
double Junior::getSalary () const  [override], [virtual]
```

Returns the junior researcher's salary.

Implements Employee.

Definition at line 64 of file rank.cpp.

6.3.3.2 show()

```
void Junior::show () const  [override], [virtual]
```

Displays junior researcher information.

Implements Employee.

Definition at line 48 of file rank.cpp.

The documentation for this class was generated from the following files:

- rank.h
- rank.cpp

6.4 Publication Class Reference

Represents a scientific publication.

```
#include <publication.h>
```

Public Member Functions

- Publication (std::string title, int citationCount)

  Constructs a Publication object.

Public Attributes

- std::string title

  Title of the publication.
- int citationCount

  Number of citations.

Friends

- std::ostream & operator<< (std::ostream &os, const Publication &pub)

  Outputs publication details to the given output stream.

6.4.1 Detailed Description

Represents a scientific publication.

This class stores information about a publication, including its title and citation count.

Definition at line 7 of file publication.h.

6.4.2 Constructor & Destructor Documentation

6.4.2.1 Publication()

```
Publication::Publication (
            std::string title,
            int citationCount)
```

Constructs a Publication object.

Parameters

title	Title of the publication.
citationCount	Number of citations.

Definition at line 4 of file publication.cpp.

6.4.3 Friends And Related Symbol Documentation

6.4.3.1 operator<<

```
std::ostream & operator<< (
            std::ostream & os,
            const Publication & pub)  [friend]
```

Outputs publication details to the given output stream.

Parameters

os	Output stream.
pub	Publication to output.

Returns

Reference to the output stream.

Definition at line 7 of file publication.cpp.

6.4.4 Member Data Documentation

6.4.4.1 citationCount

```
int Publication::citationCount
```

Number of citations.

Definition at line 10 of file publication.h.

6.4.4.2 title

```
std::string Publication::title
```

Title of the publication.

Definition at line 9 of file publication.h.

The documentation for this class was generated from the following files:

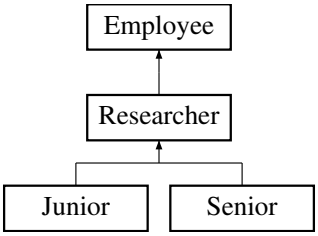
- publication.h
- publication.cpp

6.5 Researcher Class Reference

Represents a researcher employee.

```
#include <researcher.h>
```

Inheritance diagram for Researcher:



Public Member Functions

- void getPublicationsAndCitations () const
  Displays the researcher's publications and citations.
- size_t publicationCount () const
  Returns the number of publications.
- int getHirshIndex () const
  Calculates and returns the Hirsch index (h-index).
- int getMinistralPoints () const
  Returns the ministral points.
- const std::vector< Publication > & getPublications () const
  Returns a const reference to the list of publications.
- Researcher (const std::string &name, const std::string &surrname, int age, enums::Sex sex, enums::Rank rank, const FieldofResearch &field, const std::vector< Publication > &pub, int ministralPoints)
  Constructs a Researcher object.

Public Member Functions inherited from Employee

- Employee (const std::string &name, const std::string &surname, int age, enums::Sex sex, enums::Rank rank, const FieldofResearch &field)

  Constructs an Employee object.
- virtual void show () const =0

  Displays employee information (pure virtual).
- virtual double getSalary () const =0

  Returns the employee's salary (pure virtual).
- virtual ~Employee ()=default

  Virtual destructor.
- const std::string & getName () const

  Returns the employee's first name.
- const std::string & getSurname () const

  Returns the employee's surname.
- int getAge () const

  Returns the employee's age.
- enums::Sex getSex () const

  Returns the employee's sex.
- enums::Rank getRank () const

  Returns the employee's rank.
- const FieldofResearch & getFieldofresearch () const

  Returns the employee's field of research.
- const std::string sexToString () const

  Returns the employee's sex as a string.
- std::string getFieldNameAsString () const

  Returns the employee's field name as a string.
- std::string getDescriptionAsString () const

  Returns the employee's field description as a string.

Protected Attributes

- int ministralPoints

  Ministral points of the researcher.
- std::vector< Publication > publicationsAndCitations

  List of publications and their citation counts.

6.5.1 Detailed Description

Represents a researcher employee.

Inherits from Employee and adds publications and ministral points.

Definition at line 13 of file researcher.h.

6.5.2 Constructor & Destructor Documentation

6.5.2.1 Researcher()

```
Researcher::Researcher (
            const std::string & name,
            const std::string & surrname,
            int age,
            enums::Sex sex,
            enums::Rank rank,
            const FieldofResearch & field,
            const std::vector< Publication > & pub,
            int ministralPoints)
```

Constructs a Researcher object.

Parameters

name	First name.
surrname	Surname.
age	Age.
sex	Sex.
rank	Rank.
field	Field of research.
pub	List of publications.
ministralPoints	Ministral points.

Definition at line 4 of file researcher.cpp.

6.5.3 Member Function Documentation

6.5.3.1 getHirshIndex()

```
int Researcher::getHirshIndex () const
```

Calculates and returns the Hirsch index (h-index).

Definition at line 27 of file researcher.cpp.

6.5.3.2 getMinistralPoints()

```
int Researcher::getMinistralPoints () const
```

Returns the ministral points.

Definition at line 46 of file researcher.cpp.

6.5.3.3 getPublications()

```
const std::vector< Publication > & Researcher::getPublications () const
```

Returns a const reference to the list of publications.

Definition at line 51 of file researcher.cpp.

6.5.3.4 getPublicationsAndCitations()

```
void Researcher::getPublicationsAndCitations () const
```

Displays the researcher's publications and citations.

Definition at line 12 of file researcher.cpp.

6.5.3.5 publicationCount()

```
size_t Researcher::publicationCount () const
```

Returns the number of publications.

Definition at line 24 of file researcher.cpp.

6.5.4 Member Data Documentation

6.5.4.1 ministralPoints

```
int Researcher::ministralPoints  [protected]
```

Ministral points of the researcher.

Definition at line 15 of file researcher.h.

6.5.4.2 publicationsAndCitations

```
std::vector<Publication> Researcher::publicationsAndCitations  [protected]
```

List of publications and their citation counts.

Definition at line 16 of file researcher.h.

The documentation for this class was generated from the following files:

- researcher.h
- researcher.cpp

6.6 ResearcherFactory Class Reference

Factory class for creating researcher objects.

`#include <researcherFactory.h>`

Static Public Member Functions

- static std::unique_ptr< Employee > createResearcher (enums::Rank rank, const std::string &name, const std::string &surname, int age, enums::Sex sex, const FieldofResearch &field, const std::vector< Publication > &publications={}, int ministerialPoints=0, double performanceScore=0.0, int yearsOfExperience=0)

  Creates a researcher object based on the specified rank and parameters.

6.6.1 Detailed Description

Factory class for creating researcher objects.

Provides a static method to create different types of Employee-derived researchers based on the given parameters.

Definition at line 11 of file researcherFactory.h.

6.6.2 Member Function Documentation

6.6.2.1 createResearcher()

```
std::unique_ptr< Employee > ResearcherFactory::createResearcher (
            enums::Rank rank,
            const std::string & name,
            const std::string & surname,
            int age,
            enums::Sex sex,
            const FieldofResearch & field,
            const std::vector< Publication > & publications = {},
            int ministerialPoints = 0,
            double performanceScore = 0.0,
            int yearsOfExperience = 0)  [static]
```

Creates a researcher object based on the specified rank and parameters.

Depending on the rank, this method creates and returns a unique pointer to the appropriate Employee-derived object (e.g., Trainee, Junior, Senior).

Parameters

rank	Researcher's rank.
name	First name.
surname	Surname.
age	Age.
sex	Sex.
field	Field of research.
publications	List of publications (default: empty).
ministerialPoints	Ministerial points (default: 0).
performanceScore	Performance score (default: 0.0).
yearsOfExperience	Years of experience (default: 0).

Returns

Unique pointer to the created Employee object.

Definition at line 4 of file researcherFactory.cpp.

The documentation for this class was generated from the following files:

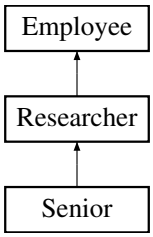
- researcherFactory.h
- researcherFactory.cpp

6.7 Senior Class Reference

Represents a senior researcher.

`#include <rank.h>`

Inheritance diagram for Senior:



Public Member Functions

- Senior (const std::string &name, const std::string &surname, int age, enums::Sex sex, enums::Rank rank, const FieldofResearch &field, const std::vector< Publication > &pub, int ministralPoints, int yearsof↩ Experience)

  Constructs a Senior object.
- int getYearsofExperience () const

  Returns the years of experience.
- void show () const override

  Displays senior researcher information.
- double getSalary () const override

  Returns the senior researcher's salary.

Public Member Functions inherited from Researcher

- void getPublicationsAndCitations () const

  Displays the researcher's publications and citations.
- size_t publicationCount () const

  Returns the number of publications.
- int getHirshIndex () const

  Calculates and returns the Hirsch index (h-index).
- int getMinistralPoints () const

  Returns the ministral points.
- const std::vector< Publication > & getPublications () const

  Returns a const reference to the list of publications.
- Researcher (const std::string &name, const std::string &surrname, int age, enums::Sex sex, enums::Rank rank, const FieldofResearch &field, const std::vector< Publication > &pub, int ministralPoints)

  Constructs a Researcher object.

Public Member Functions inherited from Employee

- Employee (const std::string &name, const std::string &surname, int age, enums::Sex sex, enums::Rank rank, const FieldofResearch &field)

  Constructs an Employee object.
- virtual ~Employee ()=default

  Virtual destructor.
- const std::string & getName () const

  Returns the employee's first name.
- const std::string & getSurname () const

  Returns the employee's surname.
- int getAge () const

  Returns the employee's age.
- enums::Sex getSex () const

  Returns the employee's sex.
- enums::Rank getRank () const

  Returns the employee's rank.
- const FieldofResearch & getFieldofresearch () const

  Returns the employee's field of research.
- const std::string sexToString () const

  Returns the employee's sex as a string.
- std::string getFieldNameAsString () const

  Returns the employee's field name as a string.
- std::string getDescriptionAsString () const

  Returns the employee's field description as a string.

Additional Inherited Members

Protected Attributes inherited from Researcher

- int ministralPoints

  Ministral points of the researcher.
- std::vector< Publication > publicationsAndCitations

  List of publications and their citation counts.

6.7.1 Detailed Description

Represents a senior researcher.

Inherits from Researcher and adds years of experience.

Definition at line 75 of file rank.h.

6.7.2 Constructor & Destructor Documentation

6.7.2.1 Senior()

```
Senior::Senior (
            const std::string & name,
            const std::string & surname,
            int age,
            enums::Sex sex,
            enums::Rank rank,
            const FieldofResearch & field,
            const std::vector< Publication > & pub,
            int ministralPoints,
            int yearsofExperience)
```

Constructs a Senior object.

Parameters

name	First name.
surname	Surname.
age	Age.
sex	Sex.
rank	Rank.
field	Field of research.
pub	List of publications.
ministralPoints	Ministral points.
yearsofExperience	Years of experience.

Definition at line 69 of file rank.cpp.

6.7.3 Member Function Documentation

6.7.3.1 getSalary()

```
double Senior::getSalary () const  [override], [virtual]
```

Returns the senior researcher's salary.

Implements Employee.

Definition at line 104 of file rank.cpp.

6.7.3.2 getYearsofExperience()

```
int Senior::getYearsofExperience () const
```

Returns the years of experience.

Definition at line 80 of file rank.cpp.

6.7.3.3 show()

```
void Senior::show () const  [override], [virtual]
```

Displays senior researcher information.

Implements Employee.

Definition at line 85 of file rank.cpp.

The documentation for this class was generated from the following files:

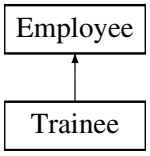
- rank.h
- rank.cpp

6.8 Trainee Class Reference

Represents a trainee employee.

```
#include <rank.h>
```

Inheritance diagram for Trainee:



Public Member Functions

- Trainee (const std::string &name, const std::string &surname, int age, enums::Sex sex, enums::Rank rank, const FieldofResearch &field, double performanceScore)

  Constructs a Trainee object.
- double getPerformanceScore () const

  Returns the performance score.
- void show () const override

  Displays trainee information.
- double getSalary () const override

  Returns the trainee's salary.

Public Member Functions inherited from Employee

- Employee (const std::string &name, const std::string &surname, int age, enums::Sex sex, enums::Rank rank, const FieldofResearch &field)

  Constructs an Employee object.
- virtual ~Employee ()=default

  Virtual destructor.
- const std::string & getName () const

  Returns the employee's first name.
- const std::string & getSurname () const

  Returns the employee's surname.
- int getAge () const

  Returns the employee's age.
- enums::Sex getSex () const

  Returns the employee's sex.
- enums::Rank getRank () const

  Returns the employee's rank.
- const FieldofResearch & getFieldofresearch () const

  Returns the employee's field of research.
- const std::string sexToString () const

  Returns the employee's sex as a string.
- std::string getFieldNameAsString () const

  Returns the employee's field name as a string.
- std::string getDescriptionAsString () const

  Returns the employee's field description as a string.

6.8.1 Detailed Description

Represents a trainee employee.

Inherits from Employee and adds a performance score.

Definition at line 12 of file rank.h.

6.8.2 Constructor & Destructor Documentation

6.8.2.1 Trainee()

```
Trainee::Trainee (
            const std::string & name,
            const std::string & surname,
            int age,
            enums::Sex sex,
            enums::Rank rank,
            const FieldofResearch & field,
            double performanceScore)
```

Constructs a Trainee object.

Parameters

name	First name.
surname	Surname.
age	Age.
sex	Sex.
rank	Rank.
field	Field of research.
performanceScore	Performance score.

Definition at line 6 of file rank.cpp.

6.8.3 Member Function Documentation

6.8.3.1 getPerformanceScore()

```
double Trainee::getPerformanceScore () const
```

Returns the performance score.

Definition at line 14 of file rank.cpp.

6.8.3.2 getSalary()

```
double Trainee::getSalary () const  [override], [virtual]
```

Returns the trainee's salary.

Implements Employee.

Definition at line 32 of file rank.cpp.

6.8.3.3 show()

```
void Trainee::show () const  [override], [virtual]
```

Displays trainee information.

Implements Employee.

Definition at line 19 of file rank.cpp.

The documentation for this class was generated from the following files:

- rank.h
- rank.cpp

Chapter 7

File Documentation

7.1 addresearcher.cpp File Reference

```
#include "addresearcher.h"
#include "publication.h"
#include "researcherFactory.h"
#include <map>
```

Functions

- void addResearcher (std::vector< std::unique_ptr< Employee > > &employees)

  Adds a new researcher to the employees collection.

7.1.1 Function Documentation

7.1.1.1 addResearcher()

```
void addResearcher (
            std::vector< std::unique_ptr< Employee > > & employees)
```

Adds a new researcher to the employees collection.

This function prompts the user for researcher details and creates a new Employee-derived object, which is then added to the provided vector.

Parameters

employees	Reference to a vector of unique pointers to Employee objects.

Definition at line 6 of file addresearcher.cpp.

7.2 addresearcher.cpp

Go to the documentation of this file.

```
00001 #include "addresearcher.h"
00002 #include "publication.h"
00003 #include "researcherFactory.h"
00004 #include <map>
00005
00006 void addResearcher(std::vector<std::unique_ptr<Employee»& employees) {
00007     std::string name, surname, fieldStr, sexStr;
00008     int age = 0;
00009     enums::Rank rank;
00010     enums::Sex sex;
00011     double performanceScore = 0.0;
00012     int ministerialPoints = 0;
00013     int yearsOfExperience = 0;
00014     std::vector<Publication> publications;
00015
00016     // ========== WYBÓR RANGI ==========
00017     std::cout « "\nWybierz range:\n";
00018     std::cout « "1. Trainee\n";
00019     std::cout « "2. Junior\n";
00020     std::cout « "3. Senior\n";
00021     std::cout « "Twoj wybor: ";
00022     int rankChoice;
00023     std::cin » rankChoice;
00024     switch (rankChoice) {
00025     case 1: rank = enums::Rank::Trainee; break;
00026     case 2: rank = enums::Rank::Junior; break;
00027     case 3: rank = enums::Rank::Senior; break;
00028     default:
00029         std::cout « "Nieprawidlowa opcja.\n";
00030         return;
00031     }
00032
00033     //dane do wprowadzenia
00034     std::cout « "Imie: "; std::cin » name;
00035     std::cout « "Nazwisko: "; std::cin » surname;
00036     std::cout « "Wiek: ";
00037     std::cin » age;
00038     if (std::cin.fail() || age < 0) {
00039         system("cls");
00040         std::cout « "Wprowadz poprawny wiek!\n";
00041         std::cin.clear();
00042         std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
00043         return;
00044
00045     }
00046
00047     std::cout « "Plec (Male/Female): "; std::cin » sexStr;
00048     if (sexStr != "Male" && sexStr != "Female") {
00049         system("cls");
00050         std::cout « "Nieprawidlowa wartosc plci.\n";
00051
00052         return;
00053
00054     }
00055     sex = (sexStr == "Male") ? enums::Sex::Male : enums::Sex::Female;
00056
00057
00058     sex = (sexStr == "Male") ? enums::Sex::Male : enums::Sex::Female;
00059
00060     std::cout « "Dziedzina naukowa (np. ComputerScience): "; std::cin » fieldStr;
00061     enums::FieldofResearchName fieldName;
00062     try {
00063         static std::map<std::string, enums::FieldofResearchName> fieldMap = {
00064             {"Physics", enums::FieldofResearchName::Physics},
00065             {"Chemistry", enums::FieldofResearchName::Chemistry},
00066             {"Biology", enums::FieldofResearchName::Biology},
00067             {"ComputerScience", enums::FieldofResearchName::ComputerScience},
00068             {"Mathematics", enums::FieldofResearchName::Mathematics},
00069             {"Medicine", enums::FieldofResearchName::Medicine},
00070             {"Psychology", enums::FieldofResearchName::Psychology},
00071             {"SocialSciences", enums::FieldofResearchName::SocialSciences},
00072             {"Economics", enums::FieldofResearchName::Economics},
00073             {"Astronomy", enums::FieldofResearchName::Astronomy},
00074             {"Neuroscience", enums::FieldofResearchName::Neuroscience},
00075             {"Robotics", enums::FieldofResearchName::Robotics},
00076             {"Geology", enums::FieldofResearchName::Geology},
00077             {"Anthropology", enums::FieldofResearchName::Anthropology}
00078         };
00079         fieldName = fieldMap.at(fieldStr);
00080     }
00081     catch (...) {
00082         system("cls");
```

```
00083         std::cout « "Nieprawidlowa dziedzina.\n";
00084         return;
00085     }
00086 
00087     FieldofResearch field(fieldName, 0.0);
00088 
00089     //tylko dla trainee
00090     if (rank == enums::Rank::Trainee) {
00091         std::cout « "Punkty stazowe (performance score): ";
00092         std::cin » performanceScore;
00093         if (std::cin.fail() || performanceScore < 0) {
00094             std::cout « "Bledna wartosc punktow stazowych.\n";
00095             std::cin.clear();
00096             std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
00097             return;
00098         }
00099 
00100     }
00101     else {
00102         std::cout « "Punkty ministerialne: ";
00103         std::cin » ministerialPoints;
00104         if (std::cin.fail() || ministerialPoints < 0) {
00105             std::cout « "Bledna wartosc punktow ministerialnych.\n";
00106             std::cin.clear();
00107             std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
00108             return;
00109         }
00110 
00111         std::cout « "Liczba publikacji: ";
00112         int pubCount;
00113         std::cin » pubCount;
00114         if (std::cin.fail() || pubCount < 0) {
00115             std::cout « "Bledna liczba publikacji.\n";
00116             std::cin.clear();
00117             std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
00118             return;
00119         }
00120         std::cin.ignore();
00121 
00122         for (int i = 0; i < pubCount; ++i) {
00123             std::string title;
00124             int citations;
00125             std::cout « "  Tytul publikacji #" « i + 1 « ": ";
00126             std::cin.ignore();
00127             std::getline(std::cin, title);
00128             std::cout « "  Liczba cytowan: ";
00129             std::cin » citations;
00130             if (std::cin.fail() || citations < 0) {
00131                 std::cout « "Bledna liczba cytowan.\n";
00132                 std::cin.clear();
00133                 std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
00134                 return;
00135             }
00136             std::cin.ignore();
00137             publications.emplace_back(title, citations);
00138         }
00139 
00140         if (rank == enums::Rank::Senior) {
00141             std::cout « "Lata doswiadczenia: ";
00142             std::cin » yearsOfExperience;
00143             if (std::cin.fail() || yearsOfExperience < 0) {
00144                 std::cout « "Bledna wartosc lat doswiadczenia.\n";
00145                 std::cin.clear();
00146                 std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
00147                 return;
00148             }
00149         }
00150     }
00151 
00152 
00153         try {
00154             auto researcher = ResearcherFactory::createResearcher(rank, name, surname, age, sex,
      field,
00155                 publications, ministerialPoints,
00156                 performanceScore, yearsOfExperience);
00157             employees.push_back(std::move(researcher));
00158             std::cout « "Dodano nowego pracownika!\n";
00159         }
00160         catch (const std::exception& e) {
00161             std::cerr « "Blad podczas tworzenia pracownika: " « e.what() « '\n';
00162         }
00163     }
```

7.3 addresearcher.h File Reference

```
#include "employee.h"
#include <vector>
#include <iostream>
```

Functions

- void addResearcher (std::vector< std::unique_ptr< Employee > > &employees)

  Adds a new researcher to the employees collection.

7.3.1 Function Documentation

7.3.1.1 addResearcher()

```
void addResearcher (
            std::vector< std::unique_ptr< Employee > > & employees)
```

Adds a new researcher to the employees collection.

This function prompts the user for researcher details and creates a new Employee-derived object, which is then added to the provided vector.

Parameters

employees	Reference to a vector of unique pointers to Employee objects.

Definition at line 6 of file addresearcher.cpp.

7.4 addresearcher.h

Go to the documentation of this file.

```
00001 #pragma once
00002 #include "employee.h"
00003 #include <vector>
00004 #include <iostream>
00005
00011 void addResearcher(std::vector<std::unique_ptr<Employee»& employees);
```

7.5 deleteresearcher.cpp File Reference

```
#include "deleteresearcher.h"
```

Functions

- void deleteResearcher (std::vector< std::unique_ptr< Employee > > &employees)

  Removes a researcher from the employees collection.

7.5.1 Function Documentation

7.5.1.1 deleteResearcher()

```
void deleteResearcher (
            std::vector< std::unique_ptr< Employee > > & employees)
```

Removes a researcher from the employees collection.

This function allows the user to select and remove a researcher from the provided vector of Employee objects.

Parameters

employees	Reference to a vector of unique pointers to Employee objects.

Definition at line 4 of file deleteresearcher.cpp.

7.6 deleteresearcher.cpp

Go to the documentation of this file.

```
00001 #include "deleteresearcher.h"
00002
00003
00004 void deleteResearcher(std::vector<std::unique_ptr<Employee»& employees) {
00005     if (employees.empty()) {
00006         system("cls");
00007         std::cout « "Brak naukowcow do usuniecia.\n";
00008         return;
00009     }
00010     system("cls");
00011     std::cout « R"(
00012   _____       _                 _   _      _   _                 _
00013  / ____|     (_)               | | (_)    | | | |               | |
00014 | (___   ___ _  ___ _ __   ___| |_ _ ___| |_| |__  _   _   _| |__
00015  \___ \ / __| |/ _ \ '_ \| __| / __| __| '_ \| | | | '_ \
00016  ____) | (__| |  __/ | | | |_| \__ \ |_| | | | | |_| | | |_) |
00017 |_____/ \___|_|\___|_| |_|\__|_|___/\__|_| |_|\__,_|_.__/
00018 ------------------------------------------------------------

00019     )" « std::endl;
00020     std::cout « "\n======= Lista naukowcow =======\n";
00021     char label = 'A';
00022
00023     for (size_t i = 0; i < employees.size() && label <= 'Z'; ++i) {
00024         std::cout « "[" « label « "] "
00025             « employees[i]->getName() « " "
00026             « employees[i]->getSurname() « " ("
00027             « employees[i]->getFieldNameAsString() « ")\n";
00028         ++label;
00029     }
00030
00031     std::cout « "\nWybierz indeks naukowca do usuniecia (lub 0 aby anulowac): ";
00032     char choice;
00033     std::cin » choice;
00034     choice = toupper(choice);
00035     system("cls");
00036     if (choice == '0') return;
00037
00038     size_t index = choice - 'A';
```

```
00039     if (index >= employees.size()) {
00040         std::cout « "Nieprawidlowy wybor.\n";
00041         return;
00042     }
00043
00044     std::cout « "\nCzy na pewno chcesz usunac "
00045         « employees[index]->getName() « " "
00046         « employees[index]->getSurname()
00047         « "? (t/n): ";
00048
00049     char confirm;
00050     std::cin » confirm;
00051     if (confirm == 't' || confirm == 'T') {
00052         employees.erase(employees.begin() + index);
00053         std::cout « "Usunieto pracownika.\n";
00054     }
00055     else {
00056         std::cout « "Anulowano usuwanie.\n";
00057     }
00058     std::cout « "Nacisnij Enter, aby wrocic do menu...";
00059     std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
00060     std::cin.get();
00061
00062 }
```

7.7 deleteresearcher.h File Reference

```
#include <cctype>
#include <memory>
#include <vector>
#include "employee.h"
```

Functions

- void deleteResearcher (std::vector< std::unique_ptr< Employee > > &employees)

  Removes a researcher from the employees collection.

7.7.1 Function Documentation

7.7.1.1 deleteResearcher()

```
void deleteResearcher (
            std::vector< std::unique_ptr< Employee > > & employees)
```

Removes a researcher from the employees collection.

This function allows the user to select and remove a researcher from the provided vector of Employee objects.

Parameters

employees	Reference to a vector of unique pointers to Employee objects.

Definition at line 4 of file deleteresearcher.cpp.

7.8 deleteresearcher.h

Go to the documentation of this file.

```
00001 #pragma once
00002 #include <cctype>
00003 #include <memory>
00004 #include <vector>
00005 #include "employee.h"
00006
00012 void deleteResearcher(std::vector<std::unique_ptr<Employee»& employees);
```

7.9 employee.cpp File Reference

```
#include "employee.h"
```

7.10 employee.cpp

Go to the documentation of this file.

```
00001 #include "employee.h"
00002
00003
00004 Employee::Employee(const std::string& name,const std::string& surname,int age,enums:: Sex
      sex,enums:: Rank rank,const FieldofResearch& field)
00005     : name(name), surname(surname), age(age), sex(sex), rank(rank), field(field) {}
00006
00007 FieldofResearch::FieldofResearch(enums::FieldofResearchName fieldName, double points)
00008     : fieldName(fieldName), description(getDescription()), points(points) {};
00009
00010 //gettery dla klasy FieldofResearch:
00011 std::string FieldofResearch::getDescription() const {
00012     switch (this->fieldName) {
00013     case enums::FieldofResearchName::Physics:
00014         return "Study of matter, energy, and the fundamental forces of nature.";
00015     case enums::FieldofResearchName::Chemistry:
00016         return "Study of substances, their properties, reactions, and transformations.";
00017     case enums::FieldofResearchName::Biology:
00018         return "Study of living organisms, their structure, function, and evolution.";
00019     case enums::FieldofResearchName::ComputerScience:
00020         return "Study of computation, programming, algorithms, and data systems.";
00021     case enums::FieldofResearchName::Mathematics:
00022         return "Study of numbers, quantities, shapes, and abstract logical structures.";
00023     case enums::FieldofResearchName::Medicine:
00024         return "Study of human health, diseases, diagnostics, and treatments.";
00025     case enums::FieldofResearchName::Psychology:
00026         return "Study of human behavior, mental processes, and cognition.";
00027     case enums::FieldofResearchName::SocialSciences:
00028         return "Study of societies, human interactions, and cultural systems.";
00029     case enums::FieldofResearchName::Economics:
00030         return "Study of production, distribution, and consumption of goods and services.";
00031     case enums::FieldofResearchName::Astronomy:
00032         return "Study of celestial objects, space, and the universe as a whole.";
00033     case enums::FieldofResearchName::Neuroscience:
00034         return "Study of the nervous system, brain structure, and neural processes.";
00035     case enums::FieldofResearchName::Robotics:
00036         return "Study of intelligent machines, automation, and robot design.";
00037     case enums::FieldofResearchName::Geology:
00038         return "Study of Earth's structure, materials, and geological processes.";
00039     case enums::FieldofResearchName::Anthropology:
00040         return "Study of human societies, cultures, and biological evolution.";
00041     default:
00042         return "Unknown field of research.";
00043     }
00044 }
00045 const enums::FieldofResearchName FieldofResearch:: getFieldName() const {
00046     return fieldName;
00047 }
00048 const double FieldofResearch:: getPoints() const {
00049     return points;
00050 }
00051 std::string FieldofResearch:: getFieldNametoString() const {
```

```
00052     switch (this->fieldName)
00053     {
00054     case enums::FieldofResearchName::Physics:
00055         return "Physics";
00056     case enums::FieldofResearchName::Chemistry:
00057         return "Chemistry";
00058     case enums::FieldofResearchName::Biology:
00059         return "Biology";
00060     case enums::FieldofResearchName::ComputerScience:
00061         return "ComputerScience";
00062     case enums::FieldofResearchName::Mathematics:
00063         return "Mathematics";
00064     case enums::FieldofResearchName::Medicine:
00065         return "Medicine";
00066     case enums::FieldofResearchName::Psychology:
00067         return "Psychology";
00068     case enums::FieldofResearchName::SocialSciences:
00069         return "Social Sciences";
00070     case enums::FieldofResearchName::Economics:
00071         return "Economics";
00072     case enums::FieldofResearchName::Astronomy:
00073         return "Astronomy";
00074     case enums::FieldofResearchName::Neuroscience:
00075         return "Neuroscience";
00076     case enums::FieldofResearchName::Robotics:
00077         return "Robotics";
00078     case enums::FieldofResearchName::Geology:
00079         return "Geology";
00080     case enums::FieldofResearchName::Anthropology:
00081         return "Anthropology";
00082     default:
00083         return "Unknown";
00084     }
00085 }
00086 const  std::string Employee:: sexToString() const {
00087     if (getSex() == enums::Sex::Male) {
00088         return "Mezczyzna";
00089     }
00090     else if (getSex() == enums::Sex::Female) {
00091         return "Kobieta";
00092     }
00093     else {
00094         return "Inna";
00095     }
00096 }
00097 std::string Employee::getFieldNameAsString() const {
00098     return field.getFieldNametoString();
00099 }
00100 std::string Employee::getDescriptionAsString() const {
00101     return field.getDescription();
00102 }
00103 //gettery dla klasy Eployee:
00104 const std::string& Employee::getName() const {
00105     return name;
00106 };
00107 const std::string& Employee::getSurname() const {
00108     return surname;
00109 };
00110 int Employee::getAge() const {
00111     return age;
00112 };
00113 enums::Sex Employee::getSex() const{
00114     return sex;
00115 };
00116 enums::Rank Employee::getRank() const {
00117     return rank;
00118 };
00119 const FieldofResearch& Employee::getFieldofresearch() const {
00120     return field;
00121 };
```

7.11 employee.h File Reference

```
#include <iostream>
#include <string>
#include "enums.h"
```

Classes

- class FieldofResearch

  Represents a scientific field of research.
- class Employee

  Abstract base class representing an employee.

7.12 employee.h

Go to the documentation of this file.

```
00001 #pragma once
00002 #include <iostream>
00003 #include <string>
00004 #include "enums.h"
00005
00010 class FieldofResearch { //#INFO klasa zamknieta
00011 private:
00012    enums::FieldofResearchName fieldName;
00013    std::string description;
00014    double points;
00015 public:
00019     FieldofResearch(enums::FieldofResearchName fieldName, double points);
00020
00022     std::string getDescription() const;
00023
00025     const enums::FieldofResearchName getFieldName() const;
00026
00028     const double getPoints() const;
00029
00031     std::string getFieldNametoString() const;
00032 };
00033
00034
00038 class Employee { //#INFO klasa zakmnieta
00039 private:
00040     std::string name;
00041     std::string surname;
00042     int age;
00043     enums::Sex sex;
00044     enums::Rank rank;
00045     FieldofResearch field;
00046
00047 public:
00055     Employee(const std::string& name, const std::string& surname, int age, enums::Sex sex, enums::Rank
      rank, const FieldofResearch& field);
00056
00058     virtual void show() const = 0;
00059
00061     virtual double getSalary() const = 0;
00062
00064     virtual ~Employee() = default;
00065
00067     const std::string& getName() const;
00068
00070     const std::string& getSurname() const;
00071
00073     int getAge() const;
00074
00076     enums::Sex getSex() const;
00077
00079     enums::Rank getRank() const;
00080
00082     const FieldofResearch& getFieldofresearch() const;
00083
00085     const std::string sexToString() const;
00086
00088     std::string getFieldNameAsString() const;
00089
00091     std::string getDescriptionAsString() const;
00092 };
```

7.13 enums.h File Reference

Contains enumerations used in the scientific team application.

Namespaces

- namespace enums

Enumerations

- enum class enums::Sex { enums::Male , enums::Female }

  Enumeration for sex.
- enum class enums::Rank { enums::Trainee , enums::Junior , enums::Senior }

  Enumeration for academic rank.
- enum class enums::FieldofResearchName { enums::Physics , enums::Chemistry , enums::Biology , enums::ComputerScience , enums::Mathematics , enums::Medicine , enums::Psychology , enums::SocialSciences , enums::Economics , enums::Astronomy , enums::Neuroscience , enums::Robotics , enums::Geology , enums::Anthropology }

  Enumeration for field of research names.

7.13.1 Detailed Description

Contains enumerations used in the scientific team application.

Definition in file enums.h.

7.14 enums.h

Go to the documentation of this file.

```
00001 #pragma once
00004
00005 namespace enums {
00006
00008 enum class Sex {
00009     Male,
00010     Female
00011 };
00012
00014 enum class Rank {
00015     Trainee,
00016     Junior,
00017     Senior
00018 };
00019
00021 enum class FieldofResearchName {
00022     Physics,
00023     Chemistry,
00024     Biology,
00025     ComputerScience,
00026     Mathematics,
00027     Medicine,
00028     Psychology,
00029     SocialSciences,
00030     Economics,
00031     Astronomy,
00032     Neuroscience,
00033     Robotics,
00034     Geology,
00035     Anthropology
00036 };
00037
00038 };
```

7.15 loadFromFile.cpp File Reference

```
#include <fstream>
#include <sstream>
#include <map>
#include "researcherFactory.h"
#include "loadFromFile.h"
```

Functions

- int safeStoi (const std::string &str)
- double safeStod (const std::string &str)
- std::vector< std::unique_ptr< Employee > > loadFromFile (const std::string &filename)

  Loads employees from a file.

7.15.1 Function Documentation

7.15.1.1 loadFromFile()

```
std::vector< std::unique_ptr< Employee > > loadFromFile (
            const std::string & filename)
```

Loads employees from a file.

This function reads employee data from the specified file and creates a vector of unique pointers to Employee objects.

Parameters

filename	Name of the file to load data from.

Returns

Vector of unique pointers to Employee objects.

Definition at line 29 of file loadFromFile.cpp.

7.15.1.2 safeStod()

```
double safeStod (
            const std::string & str)
```

Definition at line 19 of file loadFromFile.cpp.

7.15.1.3 safeStoi()

```
int safeStoi (
            const std::string & str)
```

Definition at line 8 of file loadFromFile.cpp.

7.16 loadFromFile.cpp

Go to the documentation of this file.

```
00001 #include <fstream>
00002 #include <sstream>
00003 #include <map>
00004 #include "researcherFactory.h"
00005 #include "loadFromFile.h"
00006
00007 // Bezpieczna konwersja string to int
00008 int safeStoi(const std::string& str) {
00009     try {
00010         return std::stoi(str);
00011     }
00012     catch (const std::exception& e) {
00013         std::cerr « "Błąd konwersji na int: \"" « str « "\" - " « e.what() « '\n';
00014         return 0;
00015     }
00016 }
00017
00018 // Bezpieczna konwersja string to double
00019 double safeStod(const std::string& str) {
00020     try {
00021         return std::stod(str);
00022     }
00023     catch (const std::exception& e) {
00024         std::cerr « "Błąd konwersji na double: \"" « str « "\" - " « e.what() « '\n';
00025         return 0.0;
00026     }
00027 }
00028
00029 std::vector<std::unique_ptr<Employee» loadFromFile(const std::string& filename) {
00030     std::ifstream file(filename);
00031     std::vector<std::unique_ptr<Employee» employees;
00032
00033     if (!file.is_open()) {
00034         std::cerr « "Nie mozna otworzyc pliku: " « filename « '\n';
00035         return employees;
00036     }
00037
00038     std::string line;
00039     enums::Rank currentRank;
00040     std::string name, surname, genderStr, fieldStr;
00041     int age = 0;
00042     enums::Sex sex;
00043     double performanceScore = 0.0;
00044     int ministralPoints = 0;
00045     int yearsOfExperience = 0;
00046     std::vector<Publication> publications;
00047
00048     auto parseField = [](const std::string& s) -> enums::FieldofResearchName {
00049         static std::map<std::string, enums::FieldofResearchName> fieldMap = {
00050             {"Physics", enums::FieldofResearchName::Physics},
00051             {"Chemistry", enums::FieldofResearchName::Chemistry},
00052             {"Biology", enums::FieldofResearchName::Biology},
00053             {"ComputerScience", enums::FieldofResearchName::ComputerScience},
00054             {"Mathematics", enums::FieldofResearchName::Mathematics},
00055             {"Medicine", enums::FieldofResearchName::Medicine},
00056             {"Psychology", enums::FieldofResearchName::Psychology},
00057             {"SocialSciences", enums::FieldofResearchName::SocialSciences},
00058             {"Economics", enums::FieldofResearchName::Economics},
00059             {"Astronomy", enums::FieldofResearchName::Astronomy},
00060             {"Neuroscience", enums::FieldofResearchName::Neuroscience},
00061             {"Robotics", enums::FieldofResearchName::Robotics},
00062             {"Geology", enums::FieldofResearchName::Geology},
00063             {"Anthropology", enums::FieldofResearchName::Anthropology}
00064         };
00065         auto it = fieldMap.find(s);
00066         if (it != fieldMap.end()) return it->second;
00067         std::cerr « "Nieznana dziedzina: " « s « '\n';
```

```
00068         return enums::FieldofResearchName::ComputerScience;
00069         };
00070
00071     while (std::getline(file, line)) {
00072         if (line.empty()) continue;
00073
00074         if (line[0] == '#') {
00075             if (!name.empty()) {
00076                 FieldofResearch field(parseField(fieldStr), 0.0);
00077                 try {
00078                     auto emp = ResearcherFactory::createResearcher(currentRank, name, surname, age,
      sex, field, publications, ministralPoints, performanceScore, yearsOfExperience);
00079                     employees.push_back(std::move(emp));
00080                 }
00081                 catch (const std::exception& e) {
00082                     std::cerr « "Blad podczas tworzenia pracownika: " « e.what() « '\n';
00083                 }
00084
00085                 // Reset danych
00086                 name = surname = genderStr = fieldStr = "";
00087                 age = 0;
00088                 performanceScore = 0.0;
00089                 ministralPoints = 0;
00090                 yearsOfExperience = 0;
00091                 publications.clear();
00092             }
00093
00094             if (line == "#TRAINEE") currentRank = enums::Rank::Trainee;
00095             else if (line == "#JUNIOR") currentRank = enums::Rank::Junior;
00096             else if (line == "#SENIOR") currentRank = enums::Rank::Senior;
00097         }
00098         else if (line.rfind("Imie:", 0) == 0) {
00099             name = line.substr(6);
00100         }
00101         else if (line.rfind("Nazwisko:", 0) == 0) {
00102             surname = line.substr(10);
00103         }
00104         else if (line.rfind("Wiek:", 0) == 0) {
00105             std::string value = line.substr(6);
00106             if (!value.empty()) age = safeStoi(value);
00107         }
00108         else if (line.rfind("Plec:", 0) == 0) {
00109             genderStr = line.substr(6);
00110             sex = (genderStr == "Male") ? enums::Sex::Male : enums::Sex::Female;
00111         }
00112         else if (line.rfind("Dziedzina:", 0) == 0) {
00113             fieldStr = line.substr(11);
00114         }
00115         else if (line.rfind("PunktyStazowe:", 0) == 0) {
00116             std::string val = line.substr(16);
00117             if (!val.empty()) performanceScore = safeStod(val);
00118         }
00119         else if (line.rfind("PunktyMinisterialne:", 0) == 0) {
00120             std::string val = line.substr(22);
00121             if (!val.empty()) ministralPoints = safeStoi(val);
00122         }
00123         else if (line.rfind("LataDoswiadczenia:", 0) == 0) {
00124             std::string val = line.substr(20);
00125             if (!val.empty()) yearsOfExperience = safeStoi(val);
00126         }
00127         else if (line.rfind("Tytul:", 0) == 0) {
00128             std::string pubLine = line.substr(7);
00129             size_t sep = pubLine.find("|Cytowania:");
00130             if (sep != std::string::npos) {
00131                 std::string title = pubLine.substr(0, sep);
00132                 int citations = safeStoi(pubLine.substr(sep + 12));
00133                 publications.emplace_back(title, citations);
00134             }
00135             else {
00136                 std::cerr « "Nieprawidlowy format publikacji: " « pubLine « '\n';
00137             }
00138         }
00139     }
00140
00141     // ostatni naukowiec- zabezpieczenie
00142     if (!name.empty()) {
00143         FieldofResearch field(parseField(fieldStr), 0.0);
00144         try {
00145             auto emp = ResearcherFactory::createResearcher(currentRank, name, surname, age, sex,
      field, publications, ministralPoints, performanceScore, yearsOfExperience);
00146             employees.push_back(std::move(emp));
00147         }
00148         catch (const std::exception& e) {
00149             std::cerr « "Blad podczas tworzenia ostatniego pracownika: " « e.what() « '\n';
00150         }
00151     }
00152
```

```
00153     file.close();
00154     return employees;
00155 }
```

7.17 loadFromFile.h File Reference

```
#include <fstream>
#include <sstream>
#include <map>
#include "researcherFactory.h"
```

Functions

- std::vector< std::unique_ptr< Employee > > loadFromFile (const std::string &filename)

  Loads employees from a file.

7.17.1 Function Documentation

7.17.1.1 loadFromFile()

```
std::vector< std::unique_ptr< Employee > > loadFromFile (
            const std::string & filename)
```

Loads employees from a file.

This function reads employee data from the specified file and creates a vector of unique pointers to Employee objects.

Parameters

filename	Name of the file to load data from.

Returns

Vector of unique pointers to Employee objects.

Definition at line 29 of file loadFromFile.cpp.

7.18 loadFromFile.h

Go to the documentation of this file.

```
00001 #pragma once
00002 #include <fstream>
00003 #include <sstream>
00004 #include <map>
00005 #include "researcherFactory.h"
00006
00013 std::vector<std::unique_ptr<Employee» loadFromFile(const std::string& filename);
```

7.19 main.cpp File Reference

```
#include <iostream>
#include <vector>
#include <string>
#include "ResearcherFactory.h"
#include "showMainMenu.h"
#include "showFiltered.h"
#include "loadFromFile.h"
#include "showall.h"
#include "addresearcher.h"
#include "deleteresearcher.h"
#include "savetofile.h"
#include "salarycalc.h"
```

Functions

- int main ()

7.19.1 Function Documentation

7.19.1.1 main()

```
int main ()
```

Definition at line 15 of file main.cpp.

7.20 main.cpp

Go to the documentation of this file.

```
00001 #include <iostream>
00002 #include <vector>
00003 #include <string>
00004 #include "ResearcherFactory.h"
00005 #include "showMainMenu.h"
00006 #include "showFiltered.h"
00007 #include "loadFromFile.h"
00008 #include "showall.h"
00009 #include "addresearcher.h"
00010 #include "deleteresearcher.h"
00011 #include "savetofile.h"
00012 #include "salarycalc.h"
00013
00014
00015 int main() {
00016     std::vector<std::unique_ptr<Employee» employees = loadFromFile("dane.txt");
00017     bool running = true;
00018     while (running) {
00019         showMainMenu();
00020         char choice;
00021         std::cin » choice;
00022         switch (toupper(choice)) {
00023         case '1': showFiltered(employees, enums::Rank::Senior); break;
00024         case '2': showFiltered(employees, enums::Rank::Junior); break;
00025         case '3': showFiltered(employees, enums::Rank::Trainee); break;
00026         case '4': showAll(employees); break;
00027         case 'A': addResearcher(employees); break;
00028         case 'D': deleteResearcher(employees); break;
00029         case '$': showSalaryCalc(); break;
00030         case 'Q': running = false; break;
00031         default: system("cls");  std::cout « "Nieznana opcja.\n"; break;
00032         }
00033     }
00034
00035     saveToFile(employees, "dane.txt");
00036         return 0;
00037
00038 }
```

7.21 publication.cpp File Reference

```
#include "publication.h"
```

Functions

- std::ostream & operator<< (std::ostream &os, const Publication &pub)

7.21.1 Function Documentation

7.21.1.1 operator<<()

```
std::ostream & operator<< (
            std::ostream & os,
            const Publication & pub)
```

Parameters

os	Output stream.
pub	Publication to output.

Returns

Reference to the output stream.

Definition at line 7 of file publication.cpp.

7.22 publication.cpp

Go to the documentation of this file.

```
00001 #include "publication.h"
00002
00003
00004 Publication::Publication(std::string title, int citationCount) : title(title),
      citationCount(citationCount) {}
00005
00006 //przeciazenie op. "«" w klasie Publication:
00007 std::ostream& operator«(std::ostream& os, const Publication& pub) {
00008     os « "Tytul: " « pub.title « "  |  " « "Ilosc cytowan: " « pub.citationCount;
00009     return os;
00010 }
```

7.23 publication.h File Reference

```
#include <iostream>
```

Classes

- class Publication

  Represents a scientific publication.

7.24 publication.h

Go to the documentation of this file.

```
00001 #include <iostream>
00002 #pragma once
00003
00007 class Publication {
00008 public:
00009     std::string title;
00010     int citationCount;
00011
00015     Publication(std::string title, int citationCount);
00016
00021     friend std::ostream& operator«(std::ostream& os, const Publication& pub);
00022 };
```

7.25 rank.cpp File Reference

```
#include "rank.h"
```

7.26 rank.cpp

Go to the documentation of this file.

```
00001 #include "rank.h"
00002
00003
00004
00005
00006 Trainee::Trainee(const std::string& name,
00007     const std::string& surname,
00008     int age, enums::Sex sex,
00009     enums::Rank rank,
00010     const FieldofResearch& field,
00011     double performanceScore) :Employee(name, surname, age, sex, rank, field),
00012     performanceScore(performanceScore) {};
00013
00014 double Trainee::getPerformanceScore()const{
00015     return performanceScore;
00016
00017
00018 }
00019 void Trainee::show() const {
00020     std::cout « "Imie: " « getName() « '\n';
00021     std::cout « "Nazwisko: " « getSurname() « '\n';
00022     std::cout « "Status/Stopien: Stazysta\n";
00023     std::cout « "Plec: " « sexToString() « '\n';
00024     std::cout «
      "-------------------------------------------------------------------------------------------------------------"
      « '\n';
00025     std::cout « "Dziedzina badawcza: " « getFieldNameAsString() « " ~ " « getDescriptionAsString() «
      '\n';
00026     std::cout « "Punkty stazowe: " « getPerformanceScore() « '\n';
00027     std::cout « "Wynagrodzenie (*patrz kalkulator wynagrodzen): " « getSalary() « '\n';
00028     std::cout «
      "-------------------------------------------------------------------------------------------------------------"
      « '\n';
00029
00030 }
00031
00032 double Trainee:: getSalary()const {
```

```
00033     return 3000 + ((1000 * performanceScore) / 100);
00034 }
00035
00036
00037
00038
00039 Junior::Junior(const std::string& name,
00040     const std::string& surname,
00041     int age, enums::Sex sex,
00042     enums::Rank rank,
00043     const FieldofResearch& field,
00044     const std::vector<Publication>& pub,
00045     int ministralPoints) :
00046     Researcher(name, surname, age, sex, rank, field, pub, ministralPoints) {
00047 };
00048 void Junior::show() const {
00049     std::cout « "Imie: " « getName() « '\n';
00050     std::cout « "Nazwisko: " « getSurname() « '\n';
00051     std::cout « "Status/Stopien: Junior\n";
00052     std::cout « "Plec: " « sexToString() « '\n';
00053     std::cout «
    "-----------------------------------------------------------------------------------------------------------"
    « '\n';
00054     std::cout « "Dziedzina badawcza: " « getFieldNameAsString() « " ~ " « getDescriptionAsString() «
    '\n';
00055     std::cout «
    "-----------------------------------------------------------------------------------------------------------"
    « '\n';
00056     getPublicationsAndCitations();
00057     std::cout «
    "-----------------------------------------------------------------------------------------------------------"
    « '\n';
00058     std::cout « "Indeks Hirsh'a: " « getHirshIndex() « '\n';
00059     std::cout « "Punkty ministerialne: " « getMinistralPoints() « '\n';
00060     std::cout « "Wynagrodzenie (*patrz kalkulator wynagrodzen): " « getSalary() « '\n';
00061     std::cout «
    "-----------------------------------------------------------------------------------------------------------"
    « '\n';
00062 }
00063
00064 double Junior::getSalary()const {
00065     return 5500 + ((1000 * getHirshIndex()) / 10) + 1 * ministralPoints;
00066 }
00067
00068
00069 Senior::Senior(const std::string& name,
00070     const std::string& surname,
00071     int age, enums::Sex sex,
00072     enums::Rank rank,
00073     const FieldofResearch& field,
00074     const std::vector<Publication>& pub,
00075     int ministralPoints, int yearsofExperience) :
00076     Researcher(name, surname, age, sex, rank, field, pub, ministralPoints),
00077     yearsofExperience(yearsofExperience) {
00078 };
00079
00080 int Senior::getYearsofExperience() const{
00081     return yearsofExperience;
00082 };
00083
00084
00085 void Senior::show() const {
00086     std::cout « "Imie: " « getName() « '\n';
00087     std::cout « "Nazwisko: " « getSurname() « '\n';
00088     std::cout « "Status/Stopien: Senior\n";
00089     std::cout « "Plec: " « sexToString() « '\n';
00090     std::cout «
    "-----------------------------------------------------------------------------------------------------------"
    « '\n';
00091     std::cout « "Dziedzina badawcza: " « getFieldNameAsString() « " ~ " « getDescriptionAsString() «
    '\n';
00092     std::cout «
    "-----------------------------------------------------------------------------------------------------------"
    « '\n';
00093     getPublicationsAndCitations();
00094     std::cout «
    "-----------------------------------------------------------------------------------------------------------"
    « '\n';
00095     std::cout « "Indeks Hirsh'a: " « getHirshIndex() « '\n';
00096     std::cout « "Punkty ministerialne: " « getMinistralPoints() « '\n';
00097     std::cout « "Lata doswiadczenia seniorskiego: " « getYearsofExperience()« '\n';
00098     std::cout « "Wynagrodzenie (*patrz kalkulator wynagrodzen): " « getSalary() « '\n';
00099     std::cout «
    "-----------------------------------------------------------------------------------------------------------"
    « '\n';
00100
00101
```

```
00102 }
00103
00104 double Senior::getSalary()const {
00105     return 5800 + ((2000 * getHirshIndex()) / 10) + 2 * ministralPoints + 100 *
      getYearsofExperience();
00106 }
```

7.27 rank.h File Reference

```
#include "employee.h"
#include "publication.h"
#include "researcher.h"
#include <vector>
#include <algorithm>
#include <functional>
```

Classes

- class Trainee

  Represents a trainee employee.
- class Junior

  Represents a junior researcher.
- class Senior

  Represents a senior researcher.

7.28 rank.h

Go to the documentation of this file.

```
00001 #pragma once
00002 #include "employee.h"
00003 #include "publication.h"
00004 #include "researcher.h"
00005 #include <vector>
00006 #include <algorithm>
00007 #include <functional>
00008
00012 class Trainee : public Employee {
00013 private:
00014     double performanceScore;
00015 public:
00024     Trainee(const std::string& name,
00025         const std::string& surname,
00026         int age, enums::Sex sex,
00027         enums::Rank rank,
00028         const FieldofResearch& field,
00029         double performanceScore);
00030
00032     double getPerformanceScore() const;
00033
00035     void show() const override;
00036
00038     double getSalary() const override;
00039 };
00040
00041
00045 class Junior : public Researcher {
00046 public:
00056     Junior(const std::string& name,
00057         const std::string& surname,
00058         int age, enums::Sex sex,
00059         enums::Rank rank,
00060         const FieldofResearch& field,
00061         const std::vector<Publication>& pub,
```

```
00062         int ministralPoints);
00063
00065     void show() const override;
00066
00068     double getSalary() const override;
00069 };
00070
00071
00075 class Senior : public Researcher {
00076 private:
00077     int yearsofExperience;
00078 public:
00089     Senior(const std::string& name,
00090         const std::string& surname,
00091         int age, enums::Sex sex,
00092         enums::Rank rank,
00093         const FieldofResearch& field,
00094         const std::vector<Publication>& pub,
00095         int ministralPoints, int yearsofExperience);
00096
00098     int getYearsofExperience() const;
00099
00101     void show() const override;
00102
00104     double getSalary() const override;
00105 };
```

7.29 researcher.cpp File Reference

```
#include "researcher.h"
```

7.30 researcher.cpp

Go to the documentation of this file.

```
00001 #include "researcher.h"
00002
00003
00004 Researcher::Researcher(const std::string& name,
00005     const std::string& surrname,
00006     int age, enums::Sex sex,
00007     enums::Rank rank,
00008     const FieldofResearch& field,
00009     const std::vector<Publication>& pub,
00010     int ministralPoints) :Employee(name, surrname, age, sex, rank, field),
      publicationsAndCitations(pub), ministralPoints(ministralPoints) {
00011 };
00012 void Researcher::getPublicationsAndCitations() const{
00013     if (publicationsAndCitations.empty()) {
00014         std::cout « "-Brak publikacji-\n";
00015         return;
00016     }
00017     std::cout « "Publikacje pracownika i ich cytowania: \n";
00018     int index = 1;
00019     for (const auto& pub : publicationsAndCitations) {
00020         std::cout « index « ". " « pub « '\n';
00021         index++;
00022     }
00023 };
00024 size_t Researcher::publicationCount() const {
00025     return publicationsAndCitations.size();
00026 }
00027 int Researcher::getHirshIndex() const {
00028     std::vector<int>citations;
00029     for (const Publication& pub : publicationsAndCitations) {
00030         citations.push_back(pub.citationCount);
00031     }
00032     std::sort(citations.begin(), citations.end(), std::greater<int>());
00033
00034     int h = 0;
00035     size_t elementCount = citations.size();
00036     for (int i = 0; i < elementCount; i++) {
00037         if (citations[i] >= i + 1) {
00038             h = i + 1;
```

```
00039         }
00040         else {
00041             break;
00042         }
00043     }
00044     return h;
00045 }
00046 int Researcher::getMinistralPoints() const {
00047     return ministralPoints;
00048 }
00049
00050
00051 const std::vector<Publication>& Researcher::getPublications() const {
00052     return publicationsAndCitations;
00053 }
```

7.31 researcher.h File Reference

```
#include "employee.h"
#include "enums.h"
#include "publication.h"
#include <iostream>
#include <vector>
#include <algorithm>
#include <functional>
```

Classes

- class Researcher

  Represents a researcher employee.

7.32 researcher.h

Go to the documentation of this file.

```
00001 #pragma once
00002 #include "employee.h"
00003 #include "enums.h"
00004 #include "publication.h"
00005 #include <iostream>
00006 #include <vector>
00007 #include <algorithm>
00008 #include <functional>
00009
00013 class Researcher : public Employee {
00014 protected:
00015     int ministralPoints;
00016     std::vector<Publication> publicationsAndCitations;
00017
00018 public:
00020     void getPublicationsAndCitations() const;
00021
00023     size_t publicationCount() const;
00024
00026     int getHirshIndex() const;
00027
00029     int getMinistralPoints() const;
00030
00032     const std::vector<Publication>& getPublications() const;
00033
00043     Researcher(const std::string& name,
00044         const std::string& surrname,
00045         int age, enums::Sex sex,
00046         enums::Rank rank,
00047         const FieldofResearch& field,
00048         const std::vector<Publication>& pub,
00049         int ministralPoints);
00050 };
```

7.33 researcherFactory.cpp File Reference

```
#include "researcherFactory.h"
#include <stdexcept>
```

7.34 researcherFactory.cpp

Go to the documentation of this file.

```
00001 #include "researcherFactory.h"
00002 #include <stdexcept>
00003
00004 std::unique_ptr<Employee> ResearcherFactory::createResearcher(
00005     enums::Rank rank,
00006     const std::string& name,
00007     const std::string& surname,
00008     int age,
00009     enums::Sex sex,
00010     const FieldofResearch& field,
00011     const std::vector<Publication>& publications,
00012     int ministerialPoints,
00013     double performanceScore,
00014     int yearsOfExperience
00015 ) {
00016     switch (rank) {
00017     case enums::Rank::Trainee:
00018         return std::make_unique<Trainee>(name, surname, age, sex, rank, field, performanceScore);
00019
00020     case enums::Rank::Junior:
00021         return std::make_unique<Junior>(name, surname, age, sex, rank, field, publications,
      ministerialPoints);
00022
00023     case enums::Rank::Senior:
00024         return std::make_unique<Senior>(name, surname, age, sex, rank, field, publications,
      ministerialPoints, yearsOfExperience);
00025
00026     default:
00027         throw std::invalid_argument("Invalid researcher rank.");
00028     }
00029 }
```

7.35 researcherFactory.h File Reference

```
#include "employee.h"
#include "rank.h"
#include <memory>
#include <vector>
```

Classes

- class ResearcherFactory

  Factory class for creating researcher objects.

7.36 researcherFactory.h

Go to the documentation of this file.

```
00001 #pragma once
00002 #include "employee.h"
00003 #include "rank.h"
00004 #include <memory>
00005 #include <vector>
00006
00011 class ResearcherFactory {
00012 public:
00028     static std::unique_ptr<Employee> createResearcher(
00029         enums::Rank rank,
00030         const std::string& name,
00031         const std::string& surname,
00032         int age,
00033         enums::Sex sex,
00034         const FieldofResearch& field,
00035         const std::vector<Publication>& publications = {},
00036         int ministerialPoints = 0,
00037         double performanceScore = 0.0,
00038         int yearsOfExperience = 0
00039     );
00040 };
```

7.37 salarycalc.cpp File Reference

```
#include <iostream>
#include "salarycalc.h"
```

Functions

- void showSalaryCalc ()

  Displays information about the salary calculation process.

7.37.1 Function Documentation

7.37.1.1 showSalaryCalc()

```
void showSalaryCalc ()
```

Displays information about the salary calculation process.

This function shows details or instructions related to salary calculation.

Definition at line 4 of file salarycalc.cpp.

7.38 salarycalc.cpp

Go to the documentation of this file.

```
00001 #include <iostream>
00002 #include "salarycalc.h"
00003
00004 void showSalaryCalc() {
00005     system("cls");
00006     std::cout « R"(
00007    _____       _             _   _     _   _           _
00008   / ____|     (_)           | | (_)   | | | |         | |
00009  | (___   ___ _  ___ _ __   | |_ _ ___| |_| |__   _   _| |__
00010   \___ \ / __| |/ _ \ '_ \| __| / __| __| '_ \| | | | '_ \
00011   ____) | (__| |  __/ | | | |_| \__ \ |_| | | | |_| | |_) |
00012  |_____/ \___|_|\___|_| |_|\__|_|___/\__|_| |_|\__,_|_.__/
00013 ---------------------------------------------------------

00014     )" « std::endl;
00015
00016     std::cout « "Wynagrodzenie dla stazysty: 3000 + ((1000 * <punkty stazysty>) / 100)\n";
00017     std::cout «
      "-------------------------------------------------------------------------------------------------------------------\n
00018     std::cout « "Wynagrodzenie dla juniora: 5500 + ((1000 * <indeks Hirsh'a>) / 10) + (1 * <punkty
      ministerialne>)\n";
00019     std::cout «
      "-------------------------------------------------------------------------------------------------------------------\n
00020     std::cout « "Wynagrodzenie dla seniora: 2000 * <indeks Hirsh'a>) / 10) + (2 * <punkty
      ministerialne> + 100 * <lata doswiadczenia>)\n";
00021     std::cout «
      "-------------------------------------------------------------------------------------------------------------------\n
00022
00023     std::cout « "\nNacisnij Enter, aby wrocic do menu glownego..."; //#FIX_ME do przetestowania
00024     std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
00025     std::cin.get();
00026     system("cls");
00027
00028
00029
00030
00031 }
```

7.39 salarycalc.h File Reference

Functions

- void showSalaryCalc ()

  Displays information about the salary calculation process.

7.39.1 Function Documentation

7.39.1.1 showSalaryCalc()

```
void showSalaryCalc ()
```

Displays information about the salary calculation process.

This function shows details or instructions related to salary calculation.

Definition at line 4 of file salarycalc.cpp.

7.40 salarycalc.h

Go to the documentation of this file.

```
00001 #pragma once
00002
00006 void showSalaryCalc();
```

7.41 savetofile.cpp File Reference

```
#include "savetofile.h"
#include "researcherFactory.h"
#include "researcher.h"
```

Functions

- void saveToFile (const std::vector< std::unique_ptr< Employee > > &employees, const std::string &filename)

  Saves the list of employees to a file.

7.41.1 Function Documentation

7.41.1.1 saveToFile()

```
void saveToFile (
            const std::vector< std::unique_ptr< Employee > > & employees,
            const std::string & filename)
```

Saves the list of employees to a file.

This function writes the data of all employees to the specified file.

Parameters

employees	Vector of unique pointers to Employee objects.
filename	Name of the file to save data to.

Definition at line 6 of file savetofile.cpp.

7.42 savetofile.cpp

Go to the documentation of this file.

```
00001 #include "savetofile.h"
00002 #include "researcherFactory.h"
00003 #include "researcher.h"
00004
00005
00006 void saveToFile(const std::vector<std::unique_ptr<Employee»& employees, const std::string& filename) {
00007     std::ofstream file(filename);
00008     if (!file.is_open()) {
00009         std::cerr « "Nie mozna otworzyc pliku do zapisu: " « filename « '\n';
00010         return;
00011     }
00012
00013     for (const auto& emp : employees) {
00014         enums::Rank rank = emp->getRank();
00015         switch (rank) {
00016         case enums::Rank::Trainee:
00017             file « "#TRAINEE\n";
00018             break;
00019         case enums::Rank::Junior:
00020             file « "#JUNIOR\n";
00021             break;
00022         case enums::Rank::Senior:
00023             file « "#SENIOR\n";
00024             break;
00025         default:
00026             std::cerr « "Nieznany typ rangi!\n";
00027             continue;
00028     }
00029
00030
00031         file « "Imie: " « emp->getName() « '\n';
00032         file « "Nazwisko: " « emp->getSurname() « '\n';
00033         file « "Wiek: " « emp->getAge() « '\n';
00034         file « "Plec: " « (emp->getSex() == enums::Sex::Male ? "Male" : "Female") « '\n';
00035         file « "Dziedzina: " « emp->getFieldNameAsString() « '\n';
00036
00037         if (rank == enums::Rank::Trainee) {
00038             const Trainee* t = dynamic_cast<const Trainee*>(emp.get());
00039             if (t) {
00040                 file « "PunktyStazowe: " « t->getPerformanceScore() « '\n';
00041             }
00042         }
00043
00044         if (rank == enums::Rank::Junior || rank == enums::Rank::Senior) {
00045             const Researcher* r = dynamic_cast<const Researcher*>(emp.get());
00046             if (r) {
00047                 file « "PunktyMinisterialne: " « r->getMinistralPoints() « '\n';
00048
00049                 if (rank == enums::Rank::Senior) {
00050                     const Senior* s = dynamic_cast<const Senior*>(emp.get());
00051                     if (s) {
00052                         file « "LataDoswiadczenia: " « s->getYearsofExperience() « '\n';
00053                     }
00054                 }
00055
00056                 std::vector<Publication> pubs;
00057                 if (const Junior* j = dynamic_cast<const Junior*>(r)) {
00058                     pubs = j->getPublications();
00059                 }
00060                 else if (const Senior* s = dynamic_cast<const Senior*>(r)) {
00061                     pubs = s->getPublications();
00062                 }
00063                 if (!pubs.empty()) {
00064                     file « "Publikacje:\n";
00065                     for (const auto& pub : pubs) {
00066                         file « "Tytul: " « pub.title « "|Cytowania: " « pub.citationCount « '\n';
00067                     }
00068                 }
00069             }
00070         }
00071
00072         file « '\n';
00073     }
00074
00075     std::cout « "Dane zapisane w formacie czytelnym do pliku: " « filename « '\n';
00076 }
```

7.43 savetofile.h File Reference

```
#include <fstream>
#include <vector>
#include "employee.h"
```

Functions

- void saveToFile (const std::vector< std::unique_ptr< Employee > > &employees, const std::string &filename)

  Saves the list of employees to a file.

7.43.1 Function Documentation

7.43.1.1 saveToFile()

```
void saveToFile (
            const std::vector< std::unique_ptr< Employee > > & employees,
            const std::string & filename)
```

Saves the list of employees to a file.

This function writes the data of all employees to the specified file.

Parameters

employees	Vector of unique pointers to Employee objects.
filename	Name of the file to save data to.

Definition at line 6 of file savetofile.cpp.

7.44 savetofile.h

Go to the documentation of this file.

```
00001 #pragma once
00002 #include <fstream>
00003 #include <vector>
00004 #include "employee.h"
00005
00011 void saveToFile(const std::vector<std::unique_ptr<Employee»& employees, const std::string& filename);
```

7.45 showAll.cpp File Reference

```
#include <cctype>
#include "showall.h"
```

Functions

- void showAll (const std::vector< std::unique_ptr< Employee > > &employees)

  Displays information about all employees.

7.45.1 Function Documentation

7.45.1.1 showAll()

```
void showAll (
            const std::vector< std::unique_ptr< Employee > > & employees)
```

Displays information about all employees.

This function iterates through the vector of employees and displays their details.

Parameters

employees	Vector of unique pointers to Employee objects.

Definition at line 4 of file showAll.cpp.

7.46 showAll.cpp

Go to the documentation of this file.

```
00001 #include <cctype>
00002 #include "showall.h"
00003
00004 void showAll(const std::vector<std::unique_ptr<Employee»& employees) {
00005     if (employees.empty()) {
00006         system("cls");
00007         std::cout « "Brak zapisanych naukowcow.\n";
00008         return;
00009     }
00010
00011     std::cout « "\n========= Wszyscy naukowcy =========\n";
00012     char label = 'A';
00013
00014
00015     std::vector<size_t> instances;
00016     system("cls");
00017     std::cout « R"(
00018    _____       _                 _   _      _   _             _
00019   / ____|     (_)               | | (_)    | | | |           | |
00020  | (___   ___ _  ___ _ __   ___| |_ _ ___| |_| |__  _   _  | |__
00021   \___ \ / __| |/ _ \ '_ \| __| / __| __| '_ \| | | | '_ \
00022   ____) | (__| |  __/ | | | |_| \__ \ |_| | | | |_| | |_) |
00023  |_____/ \___|_|\___|_| |_|\__|_|___/\__|_| |_|\__,_|_.__/
00024 ------------------------------------------------------------

00025     )" « std::endl;
00026     for (size_t i = 0; i < employees.size(); ++i) {
00027         std::cout « "[" « label « "] " « employees[i]->getName()
00028             « " " « employees[i]->getSurname() « " (" « employees[i]->getFieldNameAsString() « ")\n";
00029         instances.push_back(i);
00030         ++label;
00031
00032         if (label > 'Z') {
00033             std::cout « "Osiagnieto limit 26 naukowcow. Reszta nie bedzie pokazana.\n";
00034             break;
00035         }
00036     }
00037
00038     std::cout « "\nWybierz indeks, aby zobaczyc szczegoly (lub 0 aby anulowac): ";
```

```
00039     char choice;
00040     std::cin » choice;
00041     choice = toupper(choice);
00042     system("cls");
00043     if (choice == '0') return;
00044
00045     size_t selectedIndex = choice - 'A';
00046     std::cout « R"(
00047      _____       _             _   _      _   _            _
00048     / ____|     (_)           | | (_)    | | | |          | |
00049    | (___   ___ _  ___ _ __   | |_ _ ___| |_| |__  _   _  | |__
00050     \___ \ / __| |/ _ \ '_ \| __| / __| __| '_ \| | | | | '_ \
00051     ____) | (__| |  __/ | | | |_| \__ \ |_| | | | |_| | | |_) |
00052    |_____/ \___|_|\___|_| |_|\__|_|___/\__|_| |_|\__,_|_|_.__/
00053 ------------------------------------------------------------

00054     )" « std::endl;
00055     if (selectedIndex < instances.size()) {
00056         std::cout « "\n====== Szczegoly naukowca ======\n";
00057         employees[instances[selectedIndex]]->show();
00058     }
00059     else {
00060         std::cout « "Nieprawidlowy wybor.\n";
00061     }
00062     std::cout « "\nNacisnij Enter, aby wrocic do menu glownego..."; //#FIX_ME do przetestowania
00063     std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
00064     std::cin.get();
00065     system("cls");
00066 }
```

7.47 showall.h File Reference

```
#include "employee.h"
#include <cctype>
#include <vector>
#include <memory>
```

Functions

- void showAll (const std::vector< std::unique_ptr< Employee > > &employees)

  Displays information about all employees.

7.47.1 Function Documentation

7.47.1.1 showAll()

```
void showAll (
            const std::vector< std::unique_ptr< Employee > > & employees)
```

Displays information about all employees.

This function iterates through the vector of employees and displays their details.

Parameters

employees	Vector of unique pointers to Employee objects.

Definition at line 4 of file showAll.cpp.

7.48 showall.h

Go to the documentation of this file.

```
00001 #pragma once
00002 #include "employee.h"
00003 #include <cctype>
00004 #include <vector>
00005 #include <memory>
00006
00011 void showAll(const std::vector<std::unique_ptr<Employee»& employees);
```

7.49 showFiltered.cpp File Reference

7.50 showFiltered.cpp

Go to the documentation of this file.

```
00001 //Template przeniesiony do showFiltered.h
```

7.51 showFiltered.h File Reference

```
#include <iostream>
#include <vector>
#include "enums.h"
#include "employee.h"
```

Functions

- template<typename T = enums::Rank>
  void showFiltered (const std::vector< std::unique_ptr< Employee > > &employees, T filter)

  Displays employees filtered by a given criterion.

7.51.1 Function Documentation

7.51.1.1 showFiltered()

```
template<typename T = enums::Rank>
void showFiltered (
            const std::vector< std::unique_ptr< Employee > > & employees,
            T filter)
```

Displays employees filtered by a given criterion.

This function lists employees matching the provided filter (e.g., rank), allows the user to select one, and then displays detailed information.

Template Parameters

T	Type of filter (default: enums::Rank).

Parameters

employees	Vector of unique pointers to Employee objects.
filter	Value to filter employees by.

Definition at line 15 of file showFiltered.h.

7.52 showFiltered.h

Go to the documentation of this file.

```
00001 #pragma once
00002 #include <iostream>
00003 #include <vector>
00004 #include "enums.h"
00005 #include "employee.h"
00006
00014 template<typename T = enums::Rank>
00015 void showFiltered(const std::vector<std::unique_ptr<Employee»& employees, T filter) {
00016     std::vector<size_t> instances;
00017     char label = 'A';
00018     system("cls");
00019     std::cout « R"(
00020    _____      _             _   _     _   _           _
00021   / ____|    (_)           | | (_)   | | | |         | |
00022  | (___   ___ _  ___ _ __ | |_ _ ___| |_| |__  _   _| |__
00023   \___ \ / __| |/ _ \ '_ \| __| / __| __| '_ \| | | | '_ \
00024   ____) | (__| |  __/ | | | |_| \__ \ |_| | | | |_| | | |_) |
00025  |_____/ \___|_|\___|_| |_|\__|_|___/\__|_| |_|\__,_|_|_.__/
00026 ------------------------------------------------------------

00027     )" « std::endl;
00028     for (size_t i = 0; i < employees.size(); ++i) {
00029         if (employees[i]->getRank() == filter) {
00030             std::cout « "[" « label « "] " « employees[i]->getName() « " " «
      employees[i]->getSurname() « '\n';
00031             instances.push_back(i);
00032             ++label;
00033         }
00034     }
00035
00036     if (instances.empty()) {
00037         system("cls");
00038         std::cout « "Brak naukowcow tej rangi.\n";
00039         return;
00040     }
00041
00042     std::cout « "Wybierz indeks, aby pokazac szczegoly: ";
00043     char choice;
00044     std::cin » choice;
00045     choice = toupper(choice);
00046     system("cls");
00047
00048     size_t index = choice - 'A';
00049     if (index < instances.size()) {
00050         std::cout « R"(
00051    _____      _             _   _     _   _           _
00052   / ____|    (_)           | | (_)   | | | |         | |
00053  | (___   ___ _  ___ _ __ | |_ _ ___| |_| |__  _   _| |__
00054   \___ \ / __| |/ _ \ '_ \| __| / __| __| '_ \| | | | '_ \
00055   ____) | (__| |  __/ | | | |_| \__ \ |_| | | | |_| | | |_) |
00056  |_____/ \___|_|\___|_| |_|\__|_|___/\__|_| |_|\__,_|_|_.__/
00057 ------------------------------------------------------------

00058     )" « std::endl;
00059         employees[instances[index]]->show();
00060     }
00061     else {
00062         std::cout « "Nieprawidlowy wybor.\n";
```

```
00063     }
00064     std::cout « "\nNacisnij Enter, aby wrocic do menu glownego..."; //#FIX_ME do przetestowania
00065     std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
00066     std::cin.get();
00067     system("cls");
00068 }
```

7.53 showMainMenu.cpp File Reference

```
#include <iostream>
```

Functions

- void showMainMenu ()

  Displays the main menu of the application.

7.53.1 Function Documentation

7.53.1.1 showMainMenu()

```
void showMainMenu ()
```

Displays the main menu of the application.

This function shows the main menu options to the user.

Definition at line 3 of file showMainMenu.cpp.

7.54 showMainMenu.cpp

Go to the documentation of this file.

```
00001 #include <iostream>
00002
00003 void showMainMenu() {
00004     std::cout « R"(
00005    _____      _            _   _     _   _           _
00006   / ____|    (_)          | | (_)   | | | |         | |
00007  | (___   ___ _  ___ _ __ | |_ _ ___| |_| |__  _   _| |__
00008   \___ \ / __| |/ _ \ '_ \| __| / __| __| '_ \| | | | '_ \
00009   ____) | (__| |  __/ | | | |_| \__ \ |_| | | | |_| | |_) |
00010  |_____/ \___|_|\___|_| |_|\__|_|___/\__|_| |_|\__,_|_.__/
00011 ------------------------------------------------------------

00012     )" « std::endl;
00013     std::cout « "================ MENU ================\n";
00014     std::cout « "1 - Pokaz tylko Seniorow\n";
00015     std::cout « "2 - Pokaz tylko Juniorow\n";
00016     std::cout « "3 - Pokaz tylko Stazystow\n";
00017     std::cout « "4 - Pokaz wszystkich\n";
00018     std::cout « "A - Dodaj naukowca\n";
00019     std::cout « "D - Usun naukowca\n";
00020     std::cout « "$ - Pokaz kalkulator wynagrodzen\n";
00021     std::cout « "Q - Zakoncz i zapisz\n";
00022     std::cout « "======================================\n";
00023     std::cout « "Wybor: ";
00024 }
```

7.55 showMainMenu.h File Reference

Functions

- void showMainMenu ()

  Displays the main menu of the application.

7.55.1 Function Documentation

7.55.1.1 showMainMenu()

```
void showMainMenu ()
```

Displays the main menu of the application.

This function shows the main menu options to the user.

Definition at line 3 of file showMainMenu.cpp.

7.56 showMainMenu.h

Go to the documentation of this file.

```
00001 #pragma once
00002
00006 void showMainMenu();
```