

Bike Service by Rafał Jurczyk

Generated by Doxygen 1.8.17



<b>1 Hierarchical Index</b>	<b>1</b>
1.1 Class Hierarchy	1
<b>2 Class Index</b>	<b>3</b>
2.1 Class List	3
<b>3 File Index</b>	<b>5</b>
3.1 File List	5
<b>4 Class Documentation</b>	<b>7</b>
4.1 Basic_Product Class Reference	7
4.1.1 Detailed Description	7
4.1.2 Constructor & Destructor Documentation	8
4.1.2.1 Basic_Product()	8
4.1.3 Member Function Documentation	8
4.1.3.1 get_description()	8
4.1.3.2 get_name()	8
4.1.3.3 get_price()	8
4.1.3.4 show_full_name()	8
4.1.4 Member Data Documentation	8
4.1.4.1 basic_price	9
4.1.4.2 description	9
4.1.4.3 name	9
4.1.4.4 price	9
4.2 Bike Class Reference	9
4.2.1 Detailed Description	10
4.2.2 Constructor & Destructor Documentation	10
4.2.2.1 Bike() [1/2]	10
4.2.2.2 Bike() [2/2]	11
4.2.3 Member Function Documentation	11
4.2.3.1 get_bike_type()	11
4.2.3.2 get_wheel_size()	11
4.2.3.3 show_full_name()	11
4.2.4 Member Data Documentation	11
4.2.4.1 type	11
4.2.4.2 wheel_size	12
4.3 Client Class Reference	12
4.3.1 Detailed Description	12
4.3.2 Constructor & Destructor Documentation	13
4.3.2.1 Client() [1/3]	13
4.3.2.2 Client() [2/3]	13
4.3.2.3 Client() [3/3]	13
4.3.3 Member Function Documentation	13

4.3.3.1 add_event()	13
4.3.3.2 get_login()	14
4.3.3.3 get_password()	14
4.3.4 Member Data Documentation	14
4.3.4.1 login	14
4.3.4.2 password	14
4.4 Date Class Reference	14
4.4.1 Detailed Description	15
4.4.2 Constructor & Destructor Documentation	15
4.4.2.1 Date() [1/2]	15
4.4.2.2 Date() [2/2]	15
4.4.2.3 ~Date()	15
4.4.3 Member Function Documentation	15
4.4.3.1 compare_dates()	15
4.4.3.2 get_day()	16
4.4.3.3 get_month()	16
4.4.3.4 get_year()	16
4.4.4 Member Data Documentation	16
4.4.4.1 day	16
4.4.4.2 month	16
4.4.4.3 year	16
4.5 Employee Class Reference	17
4.5.1 Detailed Description	17
4.5.2 Constructor & Destructor Documentation	17
4.5.2.1 Employee() [1/2]	18
4.5.2.2 Employee() [2/2]	18
4.5.2.3 ~Employee()	18
4.5.3 Member Function Documentation	18
4.5.3.1 add_event()	18
4.5.3.2 approve_event()	18
4.5.3.3 get_employee_position()	19
4.5.3.4 get_login()	19
4.5.3.5 get_password()	19
4.5.3.6 get_salary()	19
4.5.4 Member Data Documentation	19
4.5.4.1 employee_position	19
4.5.4.2 login	19
4.5.4.3 password	20
4.5.4.4 salary	20
4.6 Event Class Reference	20
4.6.1 Detailed Description	21
4.6.2 Constructor & Destructor Documentation	21

4.6.2.1 Event()	21
4.6.2.2 ~Event()	21
4.6.3 Member Function Documentation	21
4.6.3.1 get_date()	21
4.6.3.2 get_description()	21
4.6.3.3 get_is_approved()	21
4.6.3.4 get_name()	22
4.6.3.5 set_is_approved()	22
4.6.4 Member Data Documentation	22
4.6.4.1 date	22
4.6.4.2 description	22
4.6.4.3 is_approved	22
4.6.4.4 name	22
4.7 Interface Class Reference	23
4.7.1 Detailed Description	23
4.7.2 Constructor & Destructor Documentation	23
4.7.2.1 Interface()	23
4.7.2.2 ~Interface()	24
4.7.3 Member Function Documentation	24
4.7.3.1 load_interface()	24
4.7.3.2 show_client_interface()	24
4.7.3.3 show_employee_interface()	24
4.7.3.4 show_logged_client_interface()	24
4.7.3.5 show_manager_interface()	25
4.7.4 Member Data Documentation	25
4.7.4.1 choice	25
4.7.4.2 people_database	25
4.7.4.3 storage	25
4.8 Manager Class Reference	25
4.8.1 Detailed Description	26
4.8.2 Constructor & Destructor Documentation	26
4.8.2.1 Manager() [1/2]	26
4.8.2.2 Manager() [2/2]	26
4.8.3 Member Function Documentation	26
4.8.3.1 fire_staff_member()	27
4.8.3.2 hire_employee()	27
4.8.3.3 pay_salary()	27
4.8.3.4 show_employee_list()	27
4.9 PeopleDataBase Class Reference	27
4.9.1 Detailed Description	28
4.9.2 Constructor & Destructor Documentation	28
4.9.2.1 ~PeopleDataBase()	28

4.9.3 Member Function Documentation	29
4.9.3.1 add_client()	29
4.9.3.2 add_employee()	29
4.9.3.3 add_manager()	29
4.9.3.4 client_login_form()	29
4.9.3.5 employee_login_form()	29
4.9.3.6 get_clients()	30
4.9.3.7 get_employees()	30
4.9.3.8 get_managers()	30
4.9.3.9 load_users()	30
4.9.3.10 manager_login_form()	30
4.9.3.11 save_users()	30
4.9.3.12 show_clients()	30
4.9.3.13 show_employee_list()	31
4.9.3.14 show_managers()	31
4.9.4 Member Data Documentation	31
4.9.4.1 clients	31
4.9.4.2 employees	31
4.9.4.3 managers	31
4.10 Person Class Reference	31
4.10.1 Detailed Description	32
4.10.2 Constructor & Destructor Documentation	32
4.10.2.1 Person() [1/2]	32
4.10.2.2 Person() [2/2]	33
4.10.3 Member Function Documentation	33
4.10.3.1 add_event()	33
4.10.3.2 buy_a_product_or_bike()	33
4.10.3.3 get_name()	33
4.10.3.4 get_registration_date()	33
4.10.3.5 order_a_service()	34
4.10.3.6 show_ordered_services()	34
4.10.3.7 show_registration_date()	34
4.10.4 Member Data Documentation	34
4.10.4.1 name	34
4.10.4.2 ordered_services	34
4.10.4.3 registration_date	35
4.11 Product Class Reference	35
4.11.1 Detailed Description	35
4.11.2 Constructor & Destructor Documentation	35
4.11.2.1 Product() [1/2]	36
4.11.2.2 Product() [2/2]	36
4.11.3 Member Function Documentation	36

4.11.3.1 add_quantity()	36
4.11.3.2 get_quantity()	36
4.11.3.3 show_full_name()	36
4.11.3.4 subtract_quantity()	37
4.11.4 Member Data Documentation	37
4.11.4.1 quantity	37
4.12 Service Class Reference	37
4.12.1 Detailed Description	38
4.12.2 Constructor & Destructor Documentation	38
4.12.2.1 Service()	38
4.12.3 Member Function Documentation	38
4.12.3.1 compare()	38
4.12.3.2 get_required_days()	38
4.12.3.3 show_full_name()	38
4.12.4 Member Data Documentation	39
4.12.4.1 required_days	39
4.13 Storage Class Reference	39
4.13.1 Detailed Description	40
4.13.2 Constructor & Destructor Documentation	40
4.13.2.1 ~Storage()	40
4.13.3 Member Function Documentation	40
4.13.3.1 add_event()	40
4.13.3.2 approve_event()	40
4.13.3.3 check_in_storage()	41
4.13.3.4 get_balance()	41
4.13.3.5 load_storage()	41
4.13.3.6 order_a_service()	41
4.13.3.7 save_storage()	41
4.13.3.8 set_balance()	41
4.13.3.9 show_events()	42
4.13.3.10 show_services()	42
4.13.3.11 show_storage()	42
4.13.3.12 show_unapproved_events()	42
4.13.4 Member Data Documentation	42
4.13.4.1 bikes	42
4.13.4.2 earnings	42
4.13.4.3 events	43
4.13.4.4 products	43
4.13.4.5 services	43
<b>5 File Documentation</b>	<b>45</b>
5.1 basic_product.h File Reference	45

5.2 basic_product_class.cpp File Reference . . . . .	45
5.3 bike.h File Reference . . . . .	45
5.4 bike_class.cpp File Reference . . . . .	45
5.5 client.cpp File Reference . . . . .	46
5.6 client.h File Reference . . . . .	46
5.7 date_class.cpp File Reference . . . . .	46
5.8 date_class.h File Reference . . . . .	46
5.9 employee.cpp File Reference . . . . .	46
5.10 employee.h File Reference . . . . .	46
5.11 enum.cpp File Reference . . . . .	47
5.11.1 Function Documentation . . . . .	47
5.11.1.1 EmployeePositionToString() . . . . .	47
5.11.1.2 StringToEmployeePosition() . . . . .	47
5.11.1.3 StringToType() . . . . .	48
5.11.1.4 TypeToString() . . . . .	48
5.12 enum.h File Reference . . . . .	48
5.12.1 Enumeration Type Documentation . . . . .	48
5.12.1.1 EmployeePosition . . . . .	48
5.12.1.2 Type . . . . .	49
5.12.2 Function Documentation . . . . .	49
5.12.2.1 EmployeePositionToString() . . . . .	49
5.12.2.2 StringToEmployeePosition() . . . . .	49
5.12.2.3 StringToType() . . . . .	50
5.12.2.4 TypeToString() . . . . .	50
5.13 event.cpp File Reference . . . . .	50
5.14 event.h File Reference . . . . .	50
5.15 interface.cpp File Reference . . . . .	50
5.16 interface.h File Reference . . . . .	50
5.17 main.cpp File Reference . . . . .	51
5.17.1 Function Documentation . . . . .	51
5.17.1.1 main() . . . . .	51
5.18 manager.cpp File Reference . . . . .	51
5.19 manager.h File Reference . . . . .	51
5.20 people_database.cpp File Reference . . . . .	51
5.21 people_database.h File Reference . . . . .	52
5.22 person.cpp File Reference . . . . .	52
5.23 person.h File Reference . . . . .	52
5.24 product.h File Reference . . . . .	52
5.25 product_class.cpp File Reference . . . . .	53
5.26 service.h File Reference . . . . .	53
5.27 service_class.cpp File Reference . . . . .	53
5.28 storage.cpp File Reference . . . . .	53



5.29 storage.h File Reference . . . . .	53
<b>Index</b>	<b>55</b>



# Chapter 1

## Hierarchical Index

### 1.1 Class Hierarchy

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

Basic_Product . . . . .	7
Product . . . . .	35
Bike . . . . .	9
Service . . . . .	37
Date . . . . .	14
Event . . . . .	20
Interface . . . . .	23
PeopleDataBase . . . . .	27
Person . . . . .	31
Client . . . . .	12
Employee . . . . .	17
Manager . . . . .	25
Storage . . . . .	39



## Chapter 2

# Class Index

### 2.1 Class List

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

Basic_Product	7
Bike	9
Client	12
Date	14
Employee	17
Event	20
Interface	23
Manager	25
PeopleDataBase	27
Person	31
Product	35
Service	37
Storage	39



## Chapter 3

# File Index

### 3.1 File List

Here is a list of all files with brief descriptions:

basic_product.h	45
basic_product_class.cpp	45
bike.h	45
bike_class.cpp	45
client.cpp	46
client.h	46
date_class.cpp	46
date_class.h	46
employee.cpp	46
employee.h	46
enum.cpp	47
enum.h	48
event.cpp	50
event.h	50
interface.cpp	50
interface.h	50
main.cpp	51
manager.cpp	51
manager.h	51
people_database.cpp	51
people_database.h	52
person.cpp	52
person.h	52
product.h	52
product_class.cpp	53
service.h	53
service_class.cpp	53
storage.cpp	53
storage.h	53





## Chapter 4

# Class Documentation

### 4.1 Basic\_Product Class Reference

```
#include <basic_product.h>
```

Inheritance diagram for Basic\_Product:

#### Public Member Functions

- `Basic_Product` (const std::string &name, const double price, const std::string &description)  
*Default constructor for this class. All products we create have to contain this variables.*
- virtual void `show_full_name` ()=0  
*This methods makes this class a pure virtual class so we can't make any basic products and add it to our storage.*
- std::string `get_name` ()
- double `get_price` () const
- std::string `get_description` () const

#### Protected Attributes

- std::string `name`  
*All products would have it's name that refers to unique item.*
- double `price`  
*All products would have it's price that is basicly current price including discounts.*
- double `basic_price`  
*Basic price is the original price without any discounts.*
- std::string `description`  
*All clients/employees can see what is this item with details.*

#### 4.1.1 Detailed Description

Virtual class created so I can put all of the remaining products/bikes/services in one vector. It's supposed to be a class that contains all of the basic attributes of everything that can be sold in the bike service.

## 4.1.2 Constructor & Destructor Documentation

### 4.1.2.1 Basic\_Product()

```
Basic_Product::Basic_Product (
    const std::string & name,
    const double price,
    const std::string & description )
```

Default constructor for this class. All products we create have to contain this variables.

## 4.1.3 Member Function Documentation

### 4.1.3.1 get\_description()

```
std::string Basic_Product::get_description ( ) const
```

### 4.1.3.2 get\_name()

```
std::string Basic_Product::get_name ( )
```

### 4.1.3.3 get\_price()

```
double Basic_Product::get_price ( ) const
```

### 4.1.3.4 show\_full\_name()

```
virtual void Basic_Product::show_full_name ( ) [pure virtual]
```

This methods makes this class a pure virtual class so we can't make any basic products and add it to our storage.

Implemented in [Product](#), [Bike](#), and [Service](#).

## 4.1.4 Member Data Documentation

#### 4.1.4.1 basic\_price

```
double Basic_Product::basic_price [protected]
```

Basic price is the original price without any discounts.

#### 4.1.4.2 description

```
std::string Basic_Product::description [protected]
```

All clients/employees can see what is this item with details.

#### 4.1.4.3 name

```
std::string Basic_Product::name [protected]
```

All products would have it's name that refers to unique item.

#### 4.1.4.4 price

```
double Basic_Product::price [protected]
```

All products would have it's price that is basicly current price including discounts.

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

- [basic\\_product.h](#)
- [basic\\_product\\_class.cpp](#)

## 4.2 Bike Class Reference

```
#include <bike.h>
```

Inheritance diagram for Bike:

Collaboration diagram for Bike:

## Public Member Functions

- [Bike](#) (const std::string &[name](#), const double [price](#), const std::string &[description](#), const [Type](#) [type](#), const double [wheel\\_size](#))  
*This constructor should be used when you add new bike's model to the storage - it sets this bike's quantity to 0 by default.*
- [Bike](#) (const std::string &[name](#), const double [price](#), const std::string &[description](#), const [Type](#) [type](#), const double [wheel\\_size](#), const int [quantity](#))  
*This constructor is called usually in loading database cause it automatically sets current quantity from storage.*
- virtual void [show\\_full\\_name](#) () override  
*Shows bike attributes.*
- double [get\\_wheel\\_size](#) () const
- [Type](#) [get\\_bike\\_type](#) () const

## Private Attributes

- [Type](#) [type](#)  
*Bikes have 3 unique types so client that wants for example buy city bike can search for it easier.*
- double [wheel\\_size](#)  
*Bikes should also differ in sizes. Adult should look for bigger sizes than kid's bikes.*

## Additional Inherited Members

### 4.2.1 Detailed Description

This is basicly example more advanced [Product](#) class.

### 4.2.2 Constructor & Destructor Documentation

#### 4.2.2.1 [Bike\(\)](#) [1/2]

```
Bike::Bike (
    const std::string & name,
    const double price,
    const std::string & description,
    const Type type,
    const double wheel_size )
```

This constructor should be used when you add new bike's model to the storage - it sets this bike's quantity to 0 by default.

#### 4.2.2.2 Bike() [2/2]

```
Bike::Bike (
    const std::string & name,
    const double price,
    const std::string & description,
    const Type type,
    const double wheel_size,
    const int quantity )
```

This constructor is called usually in loading database cause it automatically sets current quantity from storage.

### 4.2.3 Member Function Documentation

#### 4.2.3.1 get\_bike\_type()

```
Type Bike::get_bike_type ( ) const
```

#### 4.2.3.2 get\_wheel\_size()

```
double Bike::get_wheel_size ( ) const
```

#### 4.2.3.3 show\_full\_name()

```
void Bike::show_full_name ( ) [override], [virtual]
```

Shows bike attributes.

Reimplemented from [Product](#).

### 4.2.4 Member Data Documentation

#### 4.2.4.1 type

```
Type Bike::type [private]
```

Bikes have 3 unique types so client that wants for example buy city bike can search for it easier.

#### 4.2.4.2 wheel\_size

```
double Bike::wheel_size [private]
```

Bikes should also differ in sizes. Adult should look for bigger sizes than kid's bikes.

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

- [bike.h](#)
- [bike\\_class.cpp](#)

### 4.3 Client Class Reference

```
#include <client.h>
```

Inheritance diagram for Client:

Collaboration diagram for Client:

#### Public Member Functions

- [Client](#) ()  
*This constructor should be called only by program itself. It creates default [Client](#) with name Anonymous User. It doesn't have login and password.*
- [Client](#) (const std::string &[name](#), const std::string &[login](#), const std::string &[password](#))  
*This constructor is called when you register as a new [Client](#). It automatically sets your registration date to current day.*
- [Client](#) (const std::string &[name](#), const std::string &[login](#), const std::string &[password](#), [Date](#) &date)  
*This constructor is called while loading people\_database. It just loads existing client with all it's attributes.*
- std::string [get\\_login](#) ()
- std::string [get\\_password](#) ()
- virtual void [add\\_event](#) ([Storage](#) &storage) override  
*The [Client](#) can add event using this method, but the event still have to be confirmed by [Employee](#) to be displayed in event lists.*

#### Private Attributes

- std::string [login](#)  
*All clients - except default one - has to have unique login.*
- std::string [password](#)  
*All clients - except default one - has to also have unique password.*

#### 4.3.1 Detailed Description

Class representing our customer. Default customer is Anonymous User and it can only look what's in the store. After signing in our client can also buy stuff and add unapproved events.

## 4.3.2 Constructor & Destructor Documentation

### 4.3.2.1 Client() [1/3]

```
Client::Client ( )
```

This constructor should be called only by program itself. It creates default [Client](#) with name Anonymous User. It doesn't have login and password.

### 4.3.2.2 Client() [2/3]

```
Client::Client (
    const std::string & name,
    const std::string & login,
    const std::string & password )
```

This constructor is called when you register as a new [Client](#). It automatically sets your registration date to current day.

### 4.3.2.3 Client() [3/3]

```
Client::Client (
    const std::string & name,
    const std::string & login,
    const std::string & password,
    Date & date )
```

This constructor is called while loading people\_database. It just loads existing client with all it's attributes.

## 4.3.3 Member Function Documentation

### 4.3.3.1 add\_event()

```
void Client::add_event (
    Storage & storage ) [override], [virtual]
```

The [Client](#) can add event using this method, but the event still have to be confirmed by [Employee](#) to be displayed in event lists.

Implements [Person](#).

#### 4.3.3.2 get\_login()

```
std::string Client::get_login ( )
```

#### 4.3.3.3 get\_password()

```
std::string Client::get_password ( )
```

### 4.3.4 Member Data Documentation

#### 4.3.4.1 login

```
std::string Client::login [private]
```

All clients - except default one - has to have unique login.

#### 4.3.4.2 password

```
std::string Client::password [private]
```

All clients - except default one - has to also have unique password.

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

- [client.h](#)
- [client.cpp](#)

## 4.4 Date Class Reference

```
#include <date_class.h>
```

### Public Member Functions

- [Date](#) ()  
*Default constructor creating date same as todays date.*
- [Date](#) (const int &day, const int &month, const int &year)  
*Constructor that is used to add event date or to load dates from database.*
- bool [compare\\_dates](#) ([Date](#) date)  
*Method that returns true if our passed [Date](#) is closer date to our current time.*
- int [get\\_year](#) () const
- int [get\\_month](#) () const
- int [get\\_day](#) () const
- [~Date](#) ()



## Private Attributes

- int `day`
- int `month`
- int `year`

### 4.4.1 Detailed Description

Class representing date. Dates are used to set clients/employees registration date and when events take place.

### 4.4.2 Constructor & Destructor Documentation

#### 4.4.2.1 `Date()` [1/2]

```
Date::Date ( )
```

Default constructor creating date same as todays date.

#### 4.4.2.2 `Date()` [2/2]

```
Date::Date (
    const int & day,
    const int & month,
    const int & year )
```

Constructor that is used to add event date or to load dates from database.

#### 4.4.2.3 `~Date()`

```
Date::~~Date ( )
```

### 4.4.3 Member Function Documentation

#### 4.4.3.1 `compare_dates()`

```
bool Date::compare_dates (
    Date date )
```

Method that returns true if our passed `Date` is closer date to our current time.

#### 4.4.3.2 get\_day()

```
int Date::get_day ( ) const
```

#### 4.4.3.3 get\_month()

```
int Date::get_month ( ) const
```

#### 4.4.3.4 get\_year()

```
int Date::get_year ( ) const
```

### 4.4.4 Member Data Documentation

#### 4.4.4.1 day

```
int Date::day [private]
```

#### 4.4.4.2 month

```
int Date::month [private]
```

#### 4.4.4.3 year

```
int Date::year [private]
```

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

- [date\\_class.h](#)
- [date\\_class.cpp](#)

## 4.5 Employee Class Reference

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

Inheritance diagram for Employee:

Collaboration diagram for Employee:

### Public Member Functions

- [Employee](#) ()  
*Default constructor that create employee with only name 'Anonymous User'. It's called only by program itself.*
- [Employee](#) (const std::string &[name](#), const std::string &[login](#), const std::string &[password](#), const double [salary](#), const [EmployeePosition](#) [employee\\_position](#))  
*Constructor that creates employee with all it's attributes.*
- [EmployeePosition](#) [get\\_employee\\_position](#) () const
- double [get\\_salary](#) () const
- void [approve\\_event](#) ([Storage](#) &[storage](#))  
*As employee/manager you can approve event added by client. By approving event it can be displayed in event list to everybody.*
- std::string [get\\_login](#) () const
- std::string [get\\_password](#) () const
- virtual void [add\\_event](#) ([Storage](#) &[storage](#)) override  
*As employee/manager you can add event that is already approved.*
- [~Employee](#) ()

### Private Attributes

- std::string [login](#)  
*Employee should have unique login.*
- std::string [password](#)  
*Employee should also have unique password.*
- [EmployeePosition](#) [employee\\_position](#)  
*Employee unique attribute is it's position.*
- double [salary](#)  
*Employee unique attribute is also it's salary. It's paid to it by manager.*

#### 4.5.1 Detailed Description

[Employee](#) is person who can approve events. In future it should also accept supplies etc.

#### 4.5.2 Constructor & Destructor Documentation

#### 4.5.2.1 Employee() [1/2]

```
Employee::Employee ( )
```

Default constructor that create employee with only name 'Anonymous User'. It's called only by program itself.

#### 4.5.2.2 Employee() [2/2]

```
Employee::Employee (
    const std::string & name,
    const std::string & login,
    const std::string & password,
    const double salary,
    const EmployeePosition employee_position )
```

Constructor that creates employee with all it's attributes.

#### 4.5.2.3 ~Employee()

```
Employee::~Employee ( )
```

### 4.5.3 Member Function Documentation

#### 4.5.3.1 add\_event()

```
void Employee::add_event (
    Storage & storage ) [override], [virtual]
```

As employee/manager you can add event that is already approved.

Implements [Person](#).

#### 4.5.3.2 approve\_event()

```
void Employee::approve_event (
    Storage & storage )
```

As employee/manager you can approve event added by client. By approving event it can be displayed in event list to everybody.

#### 4.5.3.3 get\_employee\_position()

```
EmployeePosition Employee::get_employee_position ( ) const
```

#### 4.5.3.4 get\_login()

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

#### 4.5.3.5 get\_password()

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

#### 4.5.3.6 get\_salary()

```
double Employee::get_salary ( ) const
```

### 4.5.4 Member Data Documentation

#### 4.5.4.1 employee\_position

```
EmployeePosition Employee::employee_position [private]
```

[Employee](#) unique attribute is it's position.

#### 4.5.4.2 login

```
std::string Employee::login [private]
```

[Employee](#) should have unique login.

#### 4.5.4.3 password

```
std::string Employee::password [private]
```

[Employee](#) should also have unique password.

#### 4.5.4.4 salary

```
double Employee::salary [private]
```

[Employee](#) unique attribute is also it's salary. It's paid to it by manager.

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

- [employee.h](#)
- [employee.cpp](#)

## 4.6 Event Class Reference

```
#include <event.h>
```

Collaboration diagram for Event:

### Public Member Functions

- [Event](#) (std::string [name](#), std::string [description](#), [Date](#) \*[date](#), bool [is\\_approved](#))  
*Default constructor that creates 'full-grown' event.*
- [Date](#) \* [get\\_date](#) ()
- bool [get\\_is\\_approved](#) ()
- void [set\\_is\\_approved](#) ()  
*This method changes [is\\_approved](#) to true.*
- std::string [get\\_name](#) () const
- std::string [get\\_description](#) () const
- [~Event](#) ()

### Private Attributes

- std::string [name](#)  
*All events should have name that can be connected to them.*
- std::string [description](#)  
*All events should also have description so clients can read more about them.*
- bool [is\\_approved](#)  
*When [is\\_approved](#) is true, the event is displayed to every user. Only Employee/Manager can approve events added by Clients.*
- [Date](#) \* [date](#)  
*Every event should have set date when it takes place.*

### 4.6.1 Detailed Description

This class represent single event. Events can be added by everybody but client's one are unapproved.

### 4.6.2 Constructor & Destructor Documentation

#### 4.6.2.1 Event()

```
Event::Event (
    std::string name,
    std::string description,
    Date * date,
    bool is_approved = false )
```

Defalut constructor that creates 'full-grown' event.

#### 4.6.2.2 ~Event()

```
Event::~Event ( )
```

### 4.6.3 Member Function Documentation

#### 4.6.3.1 get\_date()

```
Date * Event::get_date ( )
```

#### 4.6.3.2 get\_description()

```
std::string Event::get_description ( ) const
```

#### 4.6.3.3 get\_is\_approved()

```
bool Event::get_is_approved ( )
```

#### 4.6.3.4 get\_name()

```
std::string Event::get_name ( ) const
```

#### 4.6.3.5 set\_is\_approved()

```
void Event::set_is_approved ( )
```

This method changes is\_approved to true.

### 4.6.4 Member Data Documentation

#### 4.6.4.1 date

```
Date* Event::date [private]
```

Every event should have set date when it takes place.

#### 4.6.4.2 description

```
std::string Event::description [private]
```

All events should also have description so clients can read more about them.

#### 4.6.4.3 is\_approved

```
bool Event::is_approved [private]
```

When is\_approved is true, the event is displayed to every user. Only Employee/Manager can approve events added by Clients.

#### 4.6.4.4 name

```
std::string Event::name [private]
```

All events should have name that can be connected to them.

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

- [event.h](#)
- [event.cpp](#)



## 4.7 Interface Class Reference

```
#include <interface.h>
```

Collaboration diagram for Interface:

### Public Member Functions

- [Interface](#) ()  
*Default constructor that create storage and people\_database.*
- [~Interface](#) ()  
*Destructor that calls storage's and people\_database's destructors.*
- void [load\\_interface](#) ()  
*It is called at a start of program and it loads our databases.*
- void [show\\_client\\_interface](#) ([Client](#) \*client)  
*This is UI for default Anonymous [Client](#) that isn't signed.*
- void [show\\_logged\\_client\\_interface](#) ([Client](#) \*client)  
*This is UI for signed in [Client](#).*
- void [show\\_employee\\_interface](#) ([Employee](#) \*employee)  
*This is UI for signed in [Employee](#).*
- void [show\\_manager\\_interface](#) ([Manager](#) \*manager)  
*This is UI for signed in [Manager](#).*

### Private Attributes

- [Storage](#) \* storage
- [PeopleDataBase](#) \* people\_database
- int choice

#### 4.7.1 Detailed Description

Class that is used to display text and some 'graphics' in console. What it shows depends on who we're logged as.

#### 4.7.2 Constructor & Destructor Documentation

##### 4.7.2.1 Interface()

```
Interface::Interface ( )
```

Default constructor that create storage and people\_database.

#### 4.7.2.2 ~Interface()

```
Interface::~~Interface ( )
```

Destructor that calls storage's and people\_database's destructors.

### 4.7.3 Member Function Documentation

#### 4.7.3.1 load\_interface()

```
void Interface::load_interface ( )
```

It is called at a start of program and it loads our databases.

#### 4.7.3.2 show\_client\_interface()

```
void Interface::show_client_interface (
    Client * client )
```

This is UI for default Anonymous [Client](#) that isn't signed.

#### 4.7.3.3 show\_employee\_interface()

```
void Interface::show_employee_interface (
    Employee * employee )
```

This is UI for signed in [Employee](#).

#### 4.7.3.4 show\_logged\_client\_interface()

```
void Interface::show_logged_client_interface (
    Client * client )
```

This is UI for signed in [Client](#).

#### 4.7.3.5 show\_manager\_interface()

```
void Interface::show_manager_interface (
    Manager * manager )
```

This is UI for signed in [Manager](#).

### 4.7.4 Member Data Documentation

#### 4.7.4.1 choice

```
int Interface::choice [private]
```

#### 4.7.4.2 people\_database

```
PeopleDataBase* Interface::people_database [private]
```

#### 4.7.4.3 storage

```
Storage* Interface::storage [private]
```

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

- [interface.h](#)
- [interface.cpp](#)

## 4.8 Manager Class Reference

```
#include <manager.h>
```

Inheritance diagram for Manager:

Collaboration diagram for Manager:

## Public Member Functions

- [Manager](#) ()  
*Default constructor that create manager only with name 'Anonymous User'.*
- [Manager](#) (const std::string &[name](#), const std::string &[login](#), const std::string &[password](#), double [salary](#), [EmployeePosition](#) [employee\\_position](#))  
*Constructor that creates manager with it's every attribute.*
- void [fire\\_staff\\_member](#) (std::vector< [Employee](#) \* > \*employees)  
*Deletes employee from employee\_list.*
- void [show\\_employee\\_list](#) (std::vector< [Employee](#) \* > \*employees) const  
*Display all employees.*
- void [hire\\_employee](#) (std::vector< [Employee](#) \* > \*employees)  
*Add new employee to database.*
- double [pay\\_salary](#) (std::vector< [Employee](#) \* > \*employees)  
*Checks if employee we're looking for exists and if he does it returns his salary as double.*

### 4.8.1 Detailed Description

Class representing [Manager](#) of our store. He is just a employee with more power.

### 4.8.2 Constructor & Destructor Documentation

#### 4.8.2.1 [Manager](#)() [1/2]

```
Manager::Manager ( )
```

Default constructor that create manager only with name 'Anonymous User'.

#### 4.8.2.2 [Manager](#)() [2/2]

```
Manager::Manager (
    const std::string & name,
    const std::string & login,
    const std::string & password,
    double salary,
    EmployeePosition employee_position )
```

Constructor that creates manager with it's every attribute.

### 4.8.3 Member Function Documentation

#### 4.8.3.1 fire\_staff\_member()

```
void Manager::fire_staff_member (
    std::vector< Employee * > * employees )
```

Deletes employee from employee\_list.

#### 4.8.3.2 hire\_employee()

```
void Manager::hire_employee (
    std::vector< Employee * > * employees )
```

Add new employee to database.

#### 4.8.3.3 pay\_salary()

```
double Manager::pay_salary (
    std::vector< Employee * > * employees )
```

Checks if employee we're looking for exists and if he does it returns his salary as double.

#### 4.8.3.4 show\_employee\_list()

```
void Manager::show_employee_list (
    std::vector< Employee * > * employees ) const
```

Display all employees.

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

- [manager.h](#)
- [manager.cpp](#)

## 4.9 PeopleDataBase Class Reference

```
#include <people_database.h>
```

## Public Member Functions

- void `add_client` ()  
*Add new client to database.*
- void `show_clients` () const
- void `add_employee` (`Employee *employee`)  
*Add new employee to database.*
- void `show_employee_list` () const
- void `add_manager` (`Manager *manager`)  
*Add new manager to database. I made it that it is only one manager in this store but if you want expand it everything is set up.*
- void `show_managers` () const
- `std::vector< Client * >` `get_clients` ()
- `std::vector< Employee * >` `*get_employees` ()
- `std::vector< Manager * >` `get_managers` ()
- `Client * client_login_form` (`Client *client`)  
*Display login form for client. If we exit it without signing in, it return Anonymous User which equals to displaying not logged Client UI.*
- `Employee * employee_login_form` ()  
*Display login form for employee. If we exit it without signing in, it return Anonymous User which equals to displaying not logged Client UI.*
- `Manager * manager_login_form` ()  
*Display login form for manager. If we exit it without signing in, it return Anonymous User which equals to displaying not logged Client UI.*
- void `load_users` ()  
*load every client/employee/manager to our class that represent the databse*
- void `save_users` ()  
*save every client/employee/manager in our txt file*
- `~PeopleDataBase` ()  
*Constructor that clear every client/employee/manager from it's vectors and then it deletes pointers for this objects.*

## Private Attributes

- `std::vector< Client * >` `clients`
- `std::vector< Employee * >` `employees`
- `std::vector< Manager * >` `managers`

### 4.9.1 Detailed Description

Class that represent whole users database.

### 4.9.2 Constructor & Destructor Documentation

#### 4.9.2.1 `~PeopleDataBase()`

```
PeopleDataBase::~PeopleDataBase ( )
```

Constructor that clear every client/employee/manager from it's vectors and then it deletes pointers for this objects.

### 4.9.3 Member Function Documentation

#### 4.9.3.1 add\_client()

```
void PeopleDataBase::add_client ( )
```

Add new client to database.

#### 4.9.3.2 add\_employee()

```
void PeopleDataBase::add_employee (
    Employee * employee )
```

Add new employee to database.

#### 4.9.3.3 add\_manager()

```
void PeopleDataBase::add_manager (
    Manager * manager )
```

Add new manager to database. I made it that it is only one manager in this store but if you want expand it everything is set up.

#### 4.9.3.4 client\_login\_form()

```
Client * PeopleDataBase::client_login_form (
    Client * client )
```

Display login form for client. If we exit it without signing in, it return Anonymous User which equals to displaying not logged [Client](#) UI.

#### 4.9.3.5 employee\_login\_form()

```
Employee * PeopleDataBase::employee_login_form ( )
```

Display login form for employee. If we exit it without signing in, it return Anonymous User which equals to displaying not logged [Client](#) UI.

#### 4.9.3.6 get\_clients()

```
std::vector< Client * > PeopleDataBase::get_clients ( )
```

#### 4.9.3.7 get\_employees()

```
std::vector< Employee * > * PeopleDataBase::get_employees ( )
```

#### 4.9.3.8 get\_managers()

```
std::vector< Manager * > PeopleDataBase::get_managers ( )
```

#### 4.9.3.9 load\_users()

```
void PeopleDataBase::load_users ( )
```

load every client/employee/manager to our class that represent the database

#### 4.9.3.10 manager\_login\_form()

```
Manager * PeopleDataBase::manager_login_form ( )
```

Display login form for manager. If we exit it without signing in, it return Anonymous User which equals to displaying not logged Client UI.

#### 4.9.3.11 save\_users()

```
void PeopleDataBase::save_users ( )
```

save every client/employee/manager in our txt file

#### 4.9.3.12 show\_clients()

```
void PeopleDataBase::show_clients ( ) const
```



#### 4.9.3.13 show\_employee\_list()

```
void PeopleDataBase::show_employee_list ( ) const
```

#### 4.9.3.14 show\_managers()

```
void PeopleDataBase::show_managers ( ) const
```

### 4.9.4 Member Data Documentation

#### 4.9.4.1 clients

```
std::vector<Client*> PeopleDataBase::clients [private]
```

#### 4.9.4.2 employees

```
std::vector<Employee*> PeopleDataBase::employees [private]
```

#### 4.9.4.3 managers

```
std::vector<Manager*> PeopleDataBase::managers [private]
```

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

- [people\\_database.h](#)
- [people\\_database.cpp](#)

## 4.10 Person Class Reference

```
#include <person.h>
```

Inheritance diagram for Person:

Collaboration diagram for Person:

## Public Member Functions

- [Person](#) (std::string [name](#))  
*Defalut constructor that create [Person](#) with name Anonymous User.*
- [Person](#) (const std::string &[name](#), [Date](#) &[date](#))  
*Constructor that creates user with name and date.*
- void [buy\\_a\\_product\\_or\\_bike](#) ([Storage](#) &[storage](#))  
*With this method everybody can buy products/bikes from storage.*
- void [order\\_a\\_service](#) ([Storage](#) &[storage](#))  
*With this method person can order service and if it goes succesful it gets added to [ordered\\_services](#) vector.*
- std::string [get\\_name](#) ()
- void [show\\_registration\\_date](#) ()  
*Shows when this person was registered.*
- void [show\\_ordered\\_services](#) ()  
*Display all ordered services.*
- [Date](#) [get\\_registration\\_date](#) ()
- virtual void [add\\_event](#) ([Storage](#) &[storage](#))=0  
*Vritual method that makes [Person](#) pure virtual class. All clients/employees/managers should be able to add event.*

## Private Attributes

- std::string [name](#)  
*Every person should at least have name so we can refer to it.*
- [Date](#) [registration\\_date](#)  
*Every registered person has to have registration date.*
- std::vector< std::string > [ordered\\_services](#)  
*Everybody can order service and this vector contains them.*

### 4.10.1 Detailed Description

Base class for all clients/employees/managers

### 4.10.2 Constructor & Destructor Documentation

#### 4.10.2.1 [Person\(\)](#) [1/2]

```
Person::Person (
    std::string name )
```

Defalut constructor that create [Person](#) with name Anonymous User.

#### 4.10.2.2 Person() [2/2]

```
Person::Person (
    const std::string & name,
    Date & date )
```

Constructor that creates user with name and date.

### 4.10.3 Member Function Documentation

#### 4.10.3.1 add\_event()

```
virtual void Person::add_event (
    Storage & storage ) [pure virtual]
```

Virtual method that makes [Person](#) pure virtual class. All clients/employees/managers should be able to add event.

Implemented in [Employee](#), and [Client](#).

#### 4.10.3.2 buy\_a\_product\_or\_bike()

```
void Person::buy_a_product_or_bike (
    Storage & storage )
```

With this method everybody can buy products/bikes from storage.

#### 4.10.3.3 get\_name()

```
std::string Person::get_name ( )
```

#### 4.10.3.4 get\_registration\_date()

```
Date Person::get_registration_date ( )
```

#### 4.10.3.5 order\_a\_service()

```
void Person::order_a_service (
    Storage & storage )
```

With this method person can order service and if it goes succesful it gets added to ordered\_services vector.

#### 4.10.3.6 show\_ordered\_services()

```
void Person::show_ordered_services ( )
```

Display all ordered services.

#### 4.10.3.7 show\_registration\_date()

```
void Person::show_registration_date ( )
```

Shows when this person was registered.

### 4.10.4 Member Data Documentation

#### 4.10.4.1 name

```
std::string Person::name [private]
```

Every person should at least have name so we can refer to it.

#### 4.10.4.2 ordered\_services

```
std::vector<std::string> Person::ordered_services [private]
```

Everybody can order service and this vector contains them.

#### 4.10.4.3 registration\_date

```
Date Person::registration_date [private]
```

Every registered person has to have registration date.

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

- [person.h](#)
- [person.cpp](#)

## 4.11 Product Class Reference

```
#include <product.h>
```

Inheritance diagram for Product:

Collaboration diagram for Product:

### Public Member Functions

- [Product](#) (const std::string &[name](#), const double [price](#), const std::string &[description](#))  
*Basic constructor that should be called when we want to add new product to our store. It sets quantity to 0.*
- [Product](#) (const std::string &[name](#), const double [price](#), const std::string &[description](#), const int [quantity](#))  
*Constructor that is used with loading storage from database. It sets product's quantity to it's current ammount.*
- void [add\\_quantity](#) (int amount)  
*Add amount we pass to our product/bike quantity.*
- void [subtract\\_quantity](#) (int amount)  
*Subtract amount we pass to our product/bike quantity.*
- int [get\\_quantity](#) () const
- virtual void [show\\_full\\_name](#) () override  
*Display product with all of it's attributes.*

### Private Attributes

- int [quantity](#)

### Additional Inherited Members

#### 4.11.1 Detailed Description

base product class for little stuff like tube or handlebar

#### 4.11.2 Constructor & Destructor Documentation

#### 4.11.2.1 Product() [1/2]

```
Product::Product (
    const std::string & name,
    const double price,
    const std::string & description )
```

Basic constructor that should be called when we want to add new product to our store. It sets quantity to 0.

#### 4.11.2.2 Product() [2/2]

```
Product::Product (
    const std::string & name,
    const double price,
    const std::string & description,
    const int quantity )
```

Constructor that is used with loading storage from database. It sets product's quantity to it's current ammount.

### 4.11.3 Member Function Documentation

#### 4.11.3.1 add\_quantity()

```
void Product::add_quantity (
    int amount )
```

Add amount we pass to our product/bike quantity.

#### 4.11.3.2 get\_quantity()

```
int Product::get_quantity ( ) const
```

#### 4.11.3.3 show\_full\_name()

```
void Product::show_full_name ( ) [override], [virtual]
```

Display product with all of it's attributes.

Implements [Basic\\_Product](#).

Reimplemented in [Bike](#).

#### 4.11.3.4 subtract\_quantity()

```
void Product::subtract_quantity (
    int amount )
```

Subtract amount we pass to our product/bike quantity.

### 4.11.4 Member Data Documentation

#### 4.11.4.1 quantity

```
int Product::quantity [private]
```

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

- [product.h](#)
- [product\\_class.cpp](#)

## 4.12 Service Class Reference

```
#include <service.h>
```

Inheritance diagram for Service:

Collaboration diagram for Service:

### Public Member Functions

- [Service](#) (const std::string &[name](#), const double [price](#), const std::string &[description](#), const int [required\\_days](#))  
*Default constructor that creates service with all of it's attributes.*
- virtual void [show\\_full\\_name](#) () override  
*Method that displays our service attributes.*
- bool [compare](#) ([Service](#) \*service)  
*Method that returns true if our passed service have the same name as this service.*
- int [get\\_required\\_days](#) () const

### Private Attributes

- int [required\\_days](#)  
*Services should have a time range so we know how many days it takes to complete for example repairing our bike.*

## Additional Inherited Members

### 4.12.1 Detailed Description

This is service that our bike company provide it's supposed to be almost like simple product but services needs to be done in some range of time so that's why this class is created with extra variable (and also services don't have quantity!)

### 4.12.2 Constructor & Destructor Documentation

#### 4.12.2.1 Service()

```
Service::Service (
    const std::string & name,
    const double price,
    const std::string & description,
    const int required_days )
```

Deafault constructor that creates service with all of it's attributes.

### 4.12.3 Member Function Documentation

#### 4.12.3.1 compare()

```
bool Service::compare (
    Service * service )
```

Method that returns true if our passed service have the same name as this service.

#### 4.12.3.2 get\_required\_days()

```
int Service::get_required_days ( ) const
```

#### 4.12.3.3 show\_full\_name()

```
void Service::show_full_name ( ) [override], [virtual]
```

Method that displays our service attributes.

Implements [Basic\\_Product](#).



## 4.12.4 Member Data Documentation

### 4.12.4.1 required\_days

```
int Service::required_days [private]
```

Services should have a time range so we know how many days it takes to complete for example repairing our bike.

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

- [service.h](#)
- [service\\_class.cpp](#)

## 4.13 Storage Class Reference

```
#include <storage.h>
```

### Public Member Functions

- void [show\\_storage](#) () const
- void [show\\_services](#) () const
- void [add\\_event](#) (Event \*event)  
*You can add event to store database with this method.*
- void [show\\_events](#) ()  
*Display all approved events.*
- void [show\\_unapproved\\_events](#) ()  
*Display all unapproved events.*
- void [approve\\_event](#) (std::string name)  
*This method approve event with name same as passed name.*
- double [get\\_balance](#) () const
- void [set\\_balance](#) (double ammount)  
*Every time somebody buys something our we have to pay salary to our employee we can do it with this method.*
- void [check\\_in\\_storage](#) (std::string &item)  
*If we want to buy product/bike this method checks if it's in our store and if it is it buys it (subtract quantity and add it's price to our balance)*
- bool [order\\_a\\_service](#) (std::string &service)  
*If we want to order service this method checks if it's in our store and if it is it orders it (adding to our ordered services list and add it's price to out balance)*
- void [load\\_storage](#) ()  
*load whole products/bikes/services database*
- void [save\\_storage](#) ()  
*Save whole products/bikes/services database.*
- [~Storage](#) ()  
*Destructor that deletes all products/bikes/services/events and then clear their pointers.*

## Private Attributes

- `std::vector< Product * > products`
- `std::vector< Bike * > bikes`
- `std::vector< Service * > services`
- `std::vector< Event * > events`
- `double earnings`

*Our storage contains this service earnings in 'earnings' variable.*

### 4.13.1 Detailed Description

this class is supposed to store our every product/bike/service and our shop earnings we can check if the product we're looking for is available we can check our balance and add/subtract money this is core of our shop

### 4.13.2 Constructor & Destructor Documentation

#### 4.13.2.1 ~Storage()

```
Storage::~Storage ( )
```

Destructor that deletes all products/bikes/services/events and then clear their pointers.

### 4.13.3 Member Function Documentation

#### 4.13.3.1 add\_event()

```
void Storage::add_event (
    Event * event )
```

You can add event to store database with this method.

#### 4.13.3.2 approve\_event()

```
void Storage::approve_event (
    std::string name )
```

This method approve event with name same as passed name.

#### 4.13.3.3 check\_in\_storage()

```
void Storage::check_in_storage (
    std::string & item )
```

If we want to buy product/bike this method checks if it's in our store and if it is it buys it (subtract quantity and add it's price to our balance)

#### 4.13.3.4 get\_balance()

```
double Storage::get_balance ( ) const
```

#### 4.13.3.5 load\_storage()

```
void Storage::load_storage ( )
```

load whole products/bikes/services database

#### 4.13.3.6 order\_a\_service()

```
bool Storage::order_a_service (
    std::string & service )
```

If we want to order service this method checks if it's in our store and if it is it orders it (adding to our ordered services list and add it's price to out balance)

#### 4.13.3.7 save\_storage()

```
void Storage::save_storage ( )
```

Save whole products/bikes/services database.

#### 4.13.3.8 set\_balance()

```
void Storage::set_balance (
    double ammount )
```

Every time somebody buys something our we have to pay salary to our employee we can do it with this method.

#### 4.13.3.9 show\_events()

```
void Storage::show_events ( )
```

Display all approved events.

#### 4.13.3.10 show\_services()

```
void Storage::show_services ( ) const
```

#### 4.13.3.11 show\_storage()

```
void Storage::show_storage ( ) const
```

#### 4.13.3.12 show\_unapproved\_events()

```
void Storage::show_unapproved_events ( )
```

Display all unapproved events.

### 4.13.4 Member Data Documentation

#### 4.13.4.1 bikes

```
std::vector<Bike*> Storage::bikes [private]
```

#### 4.13.4.2 earnings

```
double Storage::earnings [private]
```

Our storage contains this service earnings in 'earnings' variable.

#### 4.13.4.3 events

```
std::vector<Event*> Storage::events [private]
```

#### 4.13.4.4 products

```
std::vector<Product*> Storage::products [private]
```

#### 4.13.4.5 services

```
std::vector<Service*> Storage::services [private]
```

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

- [storage.h](#)
- [storage.cpp](#)



## Chapter 5

# File Documentation

### 5.1 basic\_product.h File Reference

```
#include "date_class.h"
#include "enum.h"
#include <iostream>
#include <string>
Include dependency graph for basic_product.h:
```

### 5.2 basic\_product\_class.cpp File Reference

```
#include "product.h"
Include dependency graph for basic_product_class.cpp:
```

### 5.3 bike.h File Reference

```
#include "date_class.h"
#include "basic_product.h"
#include "product.h"
#include "enum.h"
#include <iostream>
#include <string>
Include dependency graph for bike.h: This graph shows which files directly or indirectly include this file:
```

#### Classes

- class [Bike](#)

### 5.4 bike\_class.cpp File Reference

```
#include "bike.h"
#include "date_class.h"
#include "enum.h"
Include dependency graph for bike_class.cpp:
```

## 5.5 client.cpp File Reference

```
#include "client.h"
```

Include dependency graph for client.cpp:

## 5.6 client.h File Reference

```
#include "person.h"
```

```
#include "storage.h"
```

```
#include "event.h"
```

```
#include "date_class.h"
```

Include dependency graph for client.h: This graph shows which files directly or indirectly include this file:

### Classes

- class [Client](#)

## 5.7 date\_class.cpp File Reference

```
#include "date_class.h"
```

```
#include <ctime>
```

Include dependency graph for date\_class.cpp:

## 5.8 date\_class.h File Reference

```
#include <iostream>
```

Include dependency graph for date\_class.h: This graph shows which files directly or indirectly include this file:

### Classes

- class [Date](#)

## 5.9 employee.cpp File Reference

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

Include dependency graph for employee.cpp:

## 5.10 employee.h File Reference

```
#include "storage.h"
```

```
#include "person.h"
```

```
#include "enum.h"
```

Include dependency graph for employee.h: This graph shows which files directly or indirectly include this file:



## Classes

- class [Employee](#)

## 5.11 enum.cpp File Reference

```
#include "enum.h"
```

Include dependency graph for enum.cpp:

## Functions

- `std::string TypeToString (const Type type)`  
*Returns bikes type as string.*
- `std::string EmployeePositionToString (const EmployeePosition employee_position)`  
*Returns employee position as string.*
- `EmployeePosition StringToEmployeePosition (const std::string &employee_position)`  
*Returns employee position as type 'EmployeePosition'.*
- `Type StringToType (const std::string &type)`  
*Returns bike type as 'Type'.*

### 5.11.1 Function Documentation

#### 5.11.1.1 [EmployeePositionToString\(\)](#)

```
std::string EmployeePositionToString (  
    const EmployeePosition employee_position )
```

Returns employee position as string.

#### 5.11.1.2 [StringToEmployeePosition\(\)](#)

```
EmployeePosition StringToEmployeePosition (  
    const std::string & employee_position )
```

Returns employee position as type 'EmployeePosition'.

### 5.11.1.3 StringToType()

```
Type StringToType (
    const std::string & type )
```

Returns bike type as 'Type'.

### 5.11.1.4 TypeToString()

```
std::string TypeToString (
    const Type type )
```

Returns bikes type as string.

## 5.12 enum.h File Reference

```
#include <iostream>
```

Include dependency graph for enum.h: This graph shows which files directly or indirectly include this file:

### Enumerations

- enum [Type](#) { [mountain](#), [city](#), [universal](#) }  
*Types of bikes in store.*
- enum [EmployeePosition](#) { [warehouseman](#), [adviser](#), [service\\_technician](#), [manager](#) }  
*Availble employee positions.*

### Functions

- `std::string TypeToString (const Type type)`  
*Returns bikes type as string.*
- `std::string EmployeePositionToString (const EmployeePosition employee_position)`  
*Returns employee position as string.*
- `EmployeePosition StringToEmployeePosition (const std::string &employee_position)`  
*Returns employee position as type 'EmployeePosition'.*
- `Type StringToType (const std::string &type)`  
*Returns bike type as 'Type'.*

## 5.12.1 Enumeration Type Documentation

### 5.12.1.1 EmployeePosition

```
enum EmployeePosition
```

Availble employee positions.

**Enumerator**

warehouseman	
adviser	
service_technician	
manager	

**5.12.1.2 Type**

enum [Type](#)

Types of bikes in store.

**Enumerator**

mountain	
city	
universal	

**5.12.2 Function Documentation****5.12.2.1 EmployeePositionToString()**

```
std::string EmployeePositionToString (
    const EmployeePosition employee_position )
```

Returns employee position as string.

**5.12.2.2 StringToEmployeePosition()**

```
EmployeePosition StringToEmployeePosition (
    const std::string & employee_position )
```

Returns employee position as type 'EmployeePosition'.

### 5.12.2.3 StringToType()

```
Type StringToType (
    const std::string & type )
```

Returns bike type as 'Type'.

### 5.12.2.4 TypeToString()

```
std::string TypeToString (
    const Type type )
```

Returns bikes type as string.

## 5.13 event.cpp File Reference

```
#include "event.h"
Include dependency graph for event.cpp:
```

## 5.14 event.h File Reference

```
#include <ctime>
#include <iostream>
#include "date_class.h"
Include dependency graph for event.h: This graph shows which files directly or indirectly include this file:
```

### Classes

- class [Event](#)

## 5.15 interface.cpp File Reference

```
#include "interface.h"
#include <iomanip>
Include dependency graph for interface.cpp:
```

## 5.16 interface.h File Reference

```
#include "storage.h"
#include "people_database.h"
#include "person.h"
#include "client.h"
#include "employee.h"
#include "enum.h"
#include "manager.h"
#include <iostream>
#include <vector>
Include dependency graph for interface.h: This graph shows which files directly or indirectly include this file:
```

## Classes

- class [Interface](#)

## 5.17 main.cpp File Reference

```
#include "interface.h"
#include "client.h"
Include dependency graph for main.cpp:
```

## Functions

- int [main](#) ()

### 5.17.1 Function Documentation

#### 5.17.1.1 main()

```
int main ( )
```

## 5.18 manager.cpp File Reference

```
#include "manager.h"
Include dependency graph for manager.cpp:
```

## 5.19 manager.h File Reference

```
#include "person.h"
#include "employee.h"
Include dependency graph for manager.h: This graph shows which files directly or indirectly include this file:
```

## Classes

- class [Manager](#)

## 5.20 people\_database.cpp File Reference

```
#include "people_database.h"
Include dependency graph for people_database.cpp:
```

## 5.21 people\_database.h File Reference

```
#include "person.h"
#include "client.h"
#include "employee.h"
#include "manager.h"
#include "enum.h"
#include <iostream>
#include <vector>
#include <fstream>
```

Include dependency graph for people\_database.h: This graph shows which files directly or indirectly include this file:

### Classes

- class [PeopleDataBase](#)

## 5.22 person.cpp File Reference

```
#include "person.h"
```

Include dependency graph for person.cpp:

## 5.23 person.h File Reference

```
#include "date_class.h"
#include "storage.h"
#include <iostream>
```

Include dependency graph for person.h: This graph shows which files directly or indirectly include this file:

### Classes

- class [Person](#)

## 5.24 product.h File Reference

```
#include "date_class.h"
#include "basic_product.h"
#include "enum.h"
#include <iostream>
#include <string>
```

Include dependency graph for product.h: This graph shows which files directly or indirectly include this file:

### Classes

- class [Product](#)

## 5.25 product\_class.cpp File Reference

```
#include "product.h"
```

Include dependency graph for product\_class.cpp:

## 5.26 service.h File Reference

```
#include "date_class.h"
```

```
#include "basic_product.h"
```

```
#include <iostream>
```

```
#include <string>
```

Include dependency graph for service.h: This graph shows which files directly or indirectly include this file:

### Classes

- class [Service](#)

## 5.27 service\_class.cpp File Reference

```
#include "service.h"
```

Include dependency graph for service\_class.cpp:

## 5.28 storage.cpp File Reference

```
#include "storage.h"
```

Include dependency graph for storage.cpp:

## 5.29 storage.h File Reference

```
#include "product.h"
```

```
#include "basic_product.h"
```

```
#include "service.h"
```

```
#include "bike.h"
```

```
#include "event.h"
```

```
#include "date_class.h"
```

```
#include "enum.h"
```

```
#include <iostream>
```

```
#include <vector>
```

```
#include <fstream>
```

Include dependency graph for storage.h: This graph shows which files directly or indirectly include this file:

### Classes

- class [Storage](#)





# Index

- ~Date
  - Date, 15
- ~Employee
  - Employee, 18
- ~Event
  - Event, 21
- ~Interface
  - Interface, 23
- ~PeopleDataBase
  - PeopleDataBase, 28
- ~Storage
  - Storage, 40
- add\_client
  - PeopleDataBase, 29
- add\_employee
  - PeopleDataBase, 29
- add\_event
  - Client, 13
  - Employee, 18
  - Person, 33
  - Storage, 40
- add\_manager
  - PeopleDataBase, 29
- add\_quantity
  - Product, 36
- adviser
  - enum.h, 49
- approve\_event
  - Employee, 18
  - Storage, 40
- basic\_price
  - Basic\_Product, 8
- Basic\_Product, 7
  - basic\_price, 8
  - Basic\_Product, 8
  - description, 9
  - get\_description, 8
  - get\_name, 8
  - get\_price, 8
  - name, 9
  - price, 9
  - show\_full\_name, 8
- basic\_product.h, 45
- basic\_product\_class.cpp, 45
- Bike, 9
  - Bike, 10
  - get\_bike\_type, 11
  - get\_wheel\_size, 11
  - show\_full\_name, 11
  - type, 11
  - wheel\_size, 11
- bike.h, 45
- bike\_class.cpp, 45
- bikes
  - Storage, 42
- buy\_a\_product\_or\_bike
  - Person, 33
- check\_in\_storage
  - Storage, 40
- choice
  - Interface, 25
- city
  - enum.h, 49
- Client, 12
  - add\_event, 13
  - Client, 13
  - get\_login, 13
  - get\_password, 14
  - login, 14
  - password, 14
- client.cpp, 46
- client.h, 46
- client\_login\_form
  - PeopleDataBase, 29
- clients
  - PeopleDataBase, 31
- compare
  - Service, 38
- compare\_dates
  - Date, 15
- Date, 14
  - ~Date, 15
  - compare\_dates, 15
  - Date, 15
  - day, 16
  - get\_day, 15
  - get\_month, 16
  - get\_year, 16
  - month, 16
  - year, 16
- date
  - Event, 22
- date\_class.cpp, 46
- date\_class.h, 46
- day
  - Date, 16

- description
  - Basic\_Product, 9
  - Event, 22
- earnings
  - Storage, 42
- Employee, 17
  - ~Employee, 18
  - add\_event, 18
  - approve\_event, 18
  - Employee, 17, 18
  - employee\_position, 19
  - get\_employee\_position, 18
  - get\_login, 19
  - get\_password, 19
  - get\_salary, 19
  - login, 19
  - password, 19
  - salary, 20
- employee.cpp, 46
- employee.h, 46
- employee\_login\_form
  - PeopleDataBase, 29
- employee\_position
  - Employee, 19
- EmployeePosition
  - enum.h, 48
- EmployeePositionToString
  - enum.cpp, 47
  - enum.h, 49
- employees
  - PeopleDataBase, 31
- enum.cpp, 47
  - EmployeePositionToString, 47
  - StringToEmployeePosition, 47
  - StringToType, 47
  - TypeToString, 48
- enum.h, 48
  - adviser, 49
  - city, 49
  - EmployeePosition, 48
  - EmployeePositionToString, 49
  - manager, 49
  - mountain, 49
  - service\_technician, 49
  - StringToEmployeePosition, 49
  - StringToType, 49
  - Type, 49
  - TypeToString, 50
  - universal, 49
  - warehouseman, 49
- Event, 20
  - ~Event, 21
  - date, 22
  - description, 22
  - Event, 21
  - get\_date, 21
  - get\_description, 21
  - get\_is\_approved, 21
  - get\_name, 21
  - is\_approved, 22
  - name, 22
  - set\_is\_approved, 22
- event.cpp, 50
- event.h, 50
- events
  - Storage, 42
- fire\_staff\_member
  - Manager, 26
- get\_balance
  - Storage, 41
- get\_bike\_type
  - Bike, 11
- get\_clients
  - PeopleDataBase, 29
- get\_date
  - Event, 21
- get\_day
  - Date, 15
- get\_description
  - Basic\_Product, 8
  - Event, 21
- get\_employee\_position
  - Employee, 18
- get\_employees
  - PeopleDataBase, 30
- get\_is\_approved
  - Event, 21
- get\_login
  - Client, 13
  - Employee, 19
- get\_managers
  - PeopleDataBase, 30
- get\_month
  - Date, 16
- get\_name
  - Basic\_Product, 8
  - Event, 21
  - Person, 33
- get\_password
  - Client, 14
  - Employee, 19
- get\_price
  - Basic\_Product, 8
- get\_quantity
  - Product, 36
- get\_registration\_date
  - Person, 33
- get\_required\_days
  - Service, 38
- get\_salary
  - Employee, 19
- get\_wheel\_size
  - Bike, 11
- get\_year
  - Date, 16

- hire\_employee
  - Manager, 27
- Interface, 23
  - ~Interface, 23
  - choice, 25
  - Interface, 23
  - load\_interface, 24
  - people\_database, 25
  - show\_client\_interface, 24
  - show\_employee\_interface, 24
  - show\_logged\_client\_interface, 24
  - show\_manager\_interface, 24
  - storage, 25
- interface.cpp, 50
- interface.h, 50
- is\_approved
  - Event, 22
- load\_interface
  - Interface, 24
- load\_storage
  - Storage, 41
- load\_users
  - PeopleDataBase, 30
- login
  - Client, 14
  - Employee, 19
- main
  - main.cpp, 51
- main.cpp, 51
  - main, 51
- Manager, 25
  - fire\_staff\_member, 26
  - hire\_employee, 27
  - Manager, 26
  - pay\_salary, 27
  - show\_employee\_list, 27
- manager
  - enum.h, 49
- manager.cpp, 51
- manager.h, 51
- manager\_login\_form
  - PeopleDataBase, 30
- managers
  - PeopleDataBase, 31
- month
  - Date, 16
- mountain
  - enum.h, 49
- name
  - Basic\_Product, 9
  - Event, 22
  - Person, 34
- order\_a\_service
  - Person, 33
- Storage, 41
- ordered\_services
  - Person, 34
- password
  - Client, 14
  - Employee, 19
- pay\_salary
  - Manager, 27
- people\_database
  - Interface, 25
- people\_database.cpp, 51
- people\_database.h, 52
- PeopleDataBase, 27
  - ~PeopleDataBase, 28
  - add\_client, 29
  - add\_employee, 29
  - add\_manager, 29
  - client\_login\_form, 29
  - clients, 31
  - employee\_login\_form, 29
  - employees, 31
  - get\_clients, 29
  - get\_employees, 30
  - get\_managers, 30
  - load\_users, 30
  - manager\_login\_form, 30
  - managers, 31
  - save\_users, 30
  - show\_clients, 30
  - show\_employee\_list, 30
  - show\_managers, 31
- Person, 31
  - add\_event, 33
  - buy\_a\_product\_or\_bike, 33
  - get\_name, 33
  - get\_registration\_date, 33
  - name, 34
  - order\_a\_service, 33
  - ordered\_services, 34
  - Person, 32
  - registration\_date, 34
  - show\_ordered\_services, 34
  - show\_registration\_date, 34
- person.cpp, 52
- person.h, 52
- price
  - Basic\_Product, 9
- Product, 35
  - add\_quantity, 36
  - get\_quantity, 36
  - Product, 35, 36
  - quantity, 37
  - show\_full\_name, 36
  - subtract\_quantity, 36
- product.h, 52
- product\_class.cpp, 53
- products
  - Storage, 43

- quantity
  - Product, 37
- registration\_date
  - Person, 34
- required\_days
  - Service, 39
- salary
  - Employee, 20
- save\_storage
  - Storage, 41
- save\_users
  - PeopleDataBase, 30
- Service, 37
  - compare, 38
  - get\_required\_days, 38
  - required\_days, 39
  - Service, 38
  - show\_full\_name, 38
- service.h, 53
- service\_class.cpp, 53
- service\_technician
  - enum.h, 49
- services
  - Storage, 43
- set\_balance
  - Storage, 41
- set\_is\_approved
  - Event, 22
- show\_client\_interface
  - Interface, 24
- show\_clients
  - PeopleDataBase, 30
- show\_employee\_interface
  - Interface, 24
- show\_employee\_list
  - Manager, 27
  - PeopleDataBase, 30
- show\_events
  - Storage, 41
- show\_full\_name
  - Basic\_Product, 8
  - Bike, 11
  - Product, 36
  - Service, 38
- show\_logged\_client\_interface
  - Interface, 24
- show\_manager\_interface
  - Interface, 24
- show\_managers
  - PeopleDataBase, 31
- show\_ordered\_services
  - Person, 34
- show\_registration\_date
  - Person, 34
- show\_services
  - Storage, 42
- show\_storage
  - Storage, 42
- show\_unapproved\_events
  - Storage, 42
- Storage, 39
  - ~Storage, 40
  - add\_event, 40
  - approve\_event, 40
  - bikes, 42
  - check\_in\_storage, 40
  - earnings, 42
  - events, 42
  - get\_balance, 41
  - load\_storage, 41
  - order\_a\_service, 41
  - products, 43
  - save\_storage, 41
  - services, 43
  - set\_balance, 41
  - show\_events, 41
  - show\_services, 42
  - show\_storage, 42
  - show\_unapproved\_events, 42
- storage
  - Interface, 25
- storage.cpp, 53
- storage.h, 53
- StringToEmployeePosition
  - enum.cpp, 47
  - enum.h, 49
- StringToType
  - enum.cpp, 47
  - enum.h, 49
- subtract\_quantity
  - Product, 36
- Type
  - enum.h, 49
- type
  - Bike, 11
- TypeToString
  - enum.cpp, 48
  - enum.h, 50
- universal
  - enum.h, 49
- warehouseman
  - enum.h, 49
- wheel\_size
  - Bike, 11
- year
  - Date, 16