Bike Service by Rafał Jurczyk

Generated by Doxygen 1.8.17

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

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

Chapter 2

Class Index

2.1 Class List

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

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

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 construcor 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 construcor 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)

  Defalut 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 databse

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]
```

Vritual 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)

  Deafault 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 servces 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 avaible 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 servces 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 }

  Avaible 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
```

Avaible 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