EOOP Project - Websites

Tanya Mehra

Project description

The purpose of the program is to allow for keeping track of websites and people in the internet, as well as different types of sites, their contents, the numbers of views or „likes", people's accounts and their requests to perform various actions on the websites, based on permissions.

Functionality

- Tracking multiple websites of different types (blogs, fanpages, company websites, news portals), and people, with different kinds of access levels (guest, fan, user, admin).
- A website:
    - Manages a group of users and admins. It can create a new user based on provided person's data and handle users logging in and out.
    - Can grant admin status to non-admin users and can revoke it. (Every admin is an user as well, but not every user is an admin).
    - Manages banned IP addresses. Users with a banned IP address cannot log in or register.
    - Manages permissions and requests regarding the content. A guest or a logged-out user only has read permissions, a logged-in user has read and write permissions and can also edit their posts, and an admin always has all the permissions and can edit other users' content as well.
    - Manages the posts contained on it. The index of all the posts can be printed out with the << operator, in addition to the other website information, such as the address, the description or the creation time.
    - Keeps track of all individual guest visits, both unique and not.
- A blog:
    - Is a website where there is only one admin – the blog's sole author. The group of admins cannot be modified.
    - Blog's admin is always logged-in.
    - Only the author has write permissions, everybody else can only read.
- A news portal:
    - Is the same as a website, but allows the admins to specify a single, main headline that will be emphasized on the index page.
    - Whenever guests visit a news portal, the number of views of the post that is the main headline will increase as well.
- A fanpage:
    - Is the same as a website, but also keeps track of „likes".
    - Every individual fan can only contribute one like (or none at all) to a given fanpage.
- A company website:
    - Is a website that doesn't have any posts, but instead displays the company's information.

    - The company's information can be updated by an admin.
- A person:
    - Represents a single person connected to the internet.
    - Has a name, a physical address, an IP address and an age.
- A guest:
    - Is a person that browses the internet and visits websites but doesn't create or edit anything.
    - Can contribute toward the guest visit count of a website.
    - Can read posts on websites, contributing toward their view count as well.
- A fan:
    - Is a guest that also distributes likes to fanpages.
- An user:
    - Represents a person's account on a website.
    - Is assigned to only one single website.
    - Has an username and a password.
    - At a given time, can be logged-in or not.
    - Can read posts on websites, even those that require being logged-in (if they are logged-in of course).
    - If logged-in, can write posts on websites, fanpages and news portals.
    - If logged-in, can edit their own posts.
- An admin:
    - Is an user that has additional permissions.
    - Can write posts on any website sans a company website.
    - Can edit any posts of any user.
    - Can change the visibility of a post (whether it requires being logged-in to read).
    - Can update company's information on a company website.
- A post:
    - Has a title and the content.
    - Has the original author assigned. If the author doesn't exist anymore, it will say [deleted] instead of their name.
    - Keeps track of individual contributors that have edited the post. (Everybody can only either appear once or zero times on that list)
    - Keeps track of the initial creation time and the time of the last change.
    - Is assigned to a single website.
    - Can either be publicly visible or require being logged-in to read.
    - Keeps track of all individual views, both unique and not.
    - Can be read by a person depending on their permissions, which increases the view count.
    - Requires permissions to be created on a website.
- The internet:
    - Is a singleton (there can only be one instance).
    - Serves as an index of all the people and all the websites.
    - Keeps track of the last IP address and simplifies the registration of connected people by automatically incrementing that address and assigning it to every consecutive person.

    - Allows to search for people based on the name or the IP.
    - Allows to search for websites based on the address.
    - Allows to search for company websites based on the company name.
    - Allows to remove individual people or websites from the vectors.
- Every class has the << operator overloaded, to allow printing.

Limitations

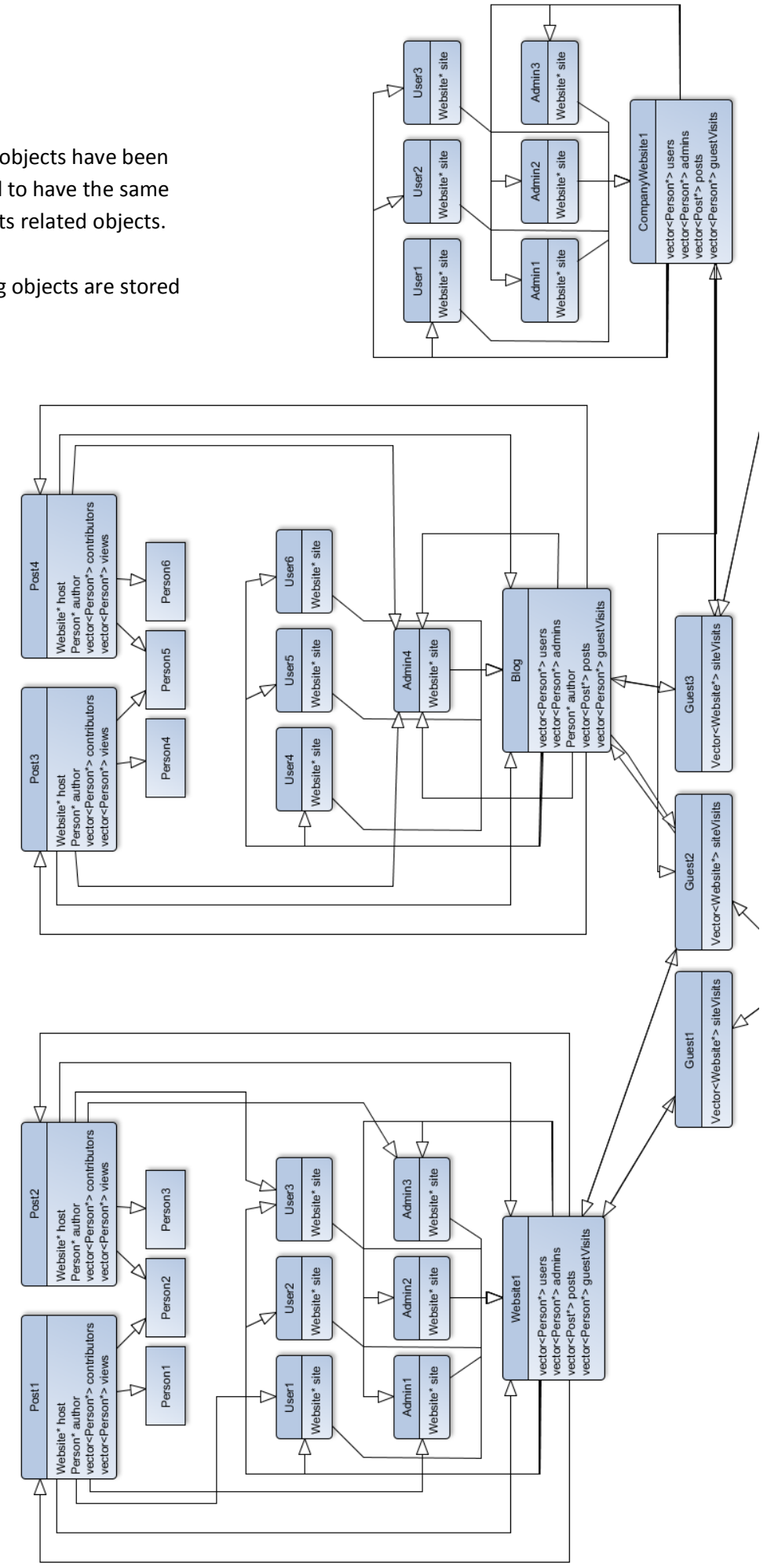
- There can only be one internet.
- Every user is assigned to only one website.
- Every fan can like an individual fanpage only once.
- There is no way to change an user's username or password.
- There is no way to change a website's address or description.
- There is no way to change a person's identity or IP address.
- There can be only one user with a given username per website.
- Objects that aren't assigned to anything and have been created dynamically must be deleted from the memory manually, as there will be no class destructor that has direct access to it.
- There is no way to create an admin account from a person in a single step – an user must be created first and then granted admin permissions.
- Clearing all user accounts from a website deletes them from the memory as well, making all their pointers unusable.

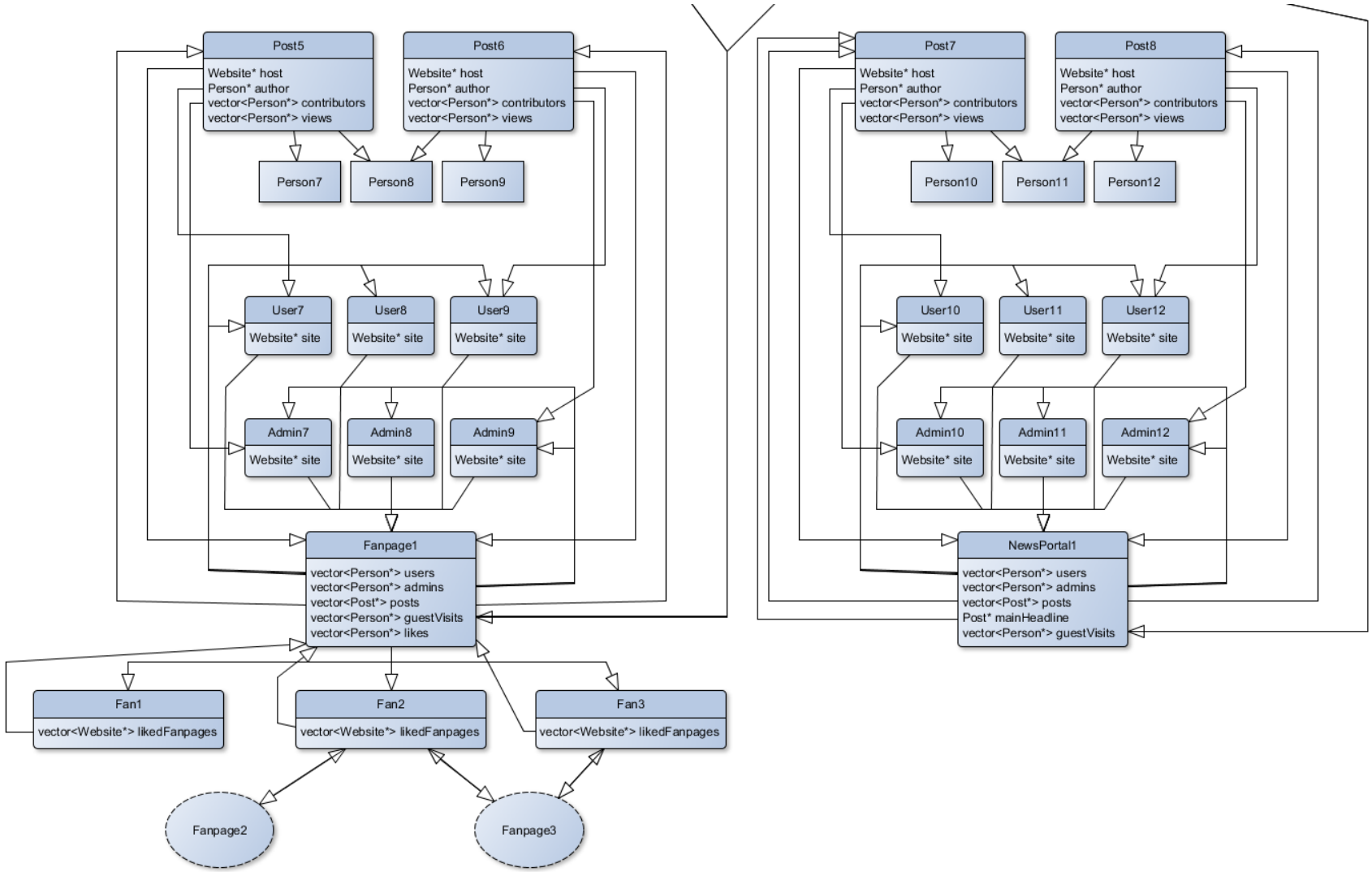
Memory map

The “Fanpage2” and “Fanpage3” objects have been simplified and are to be assumed to have the same structure as “Fanpage1” with all its related objects.

Also, the pointers to the following objects are stored in the single instance of Internet:

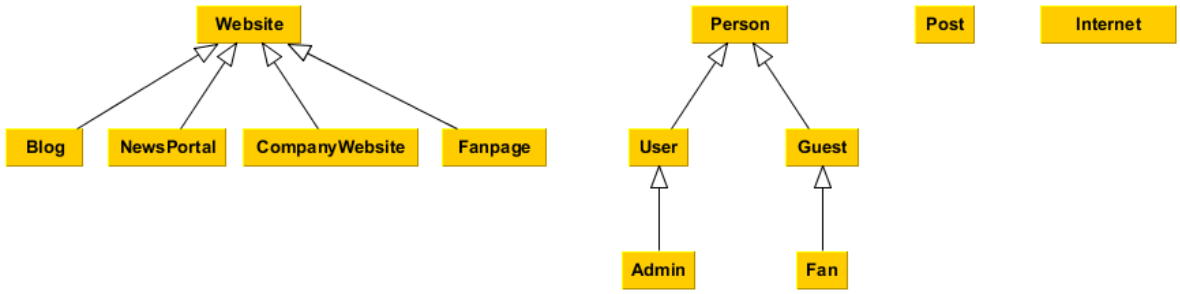
- Person#,
- User#,
- Guest#,
- Admin#,
- Fan#,
- Website1,
- Blog1,
- Fanpage#,
- CompanyWebsite1,
- NewsPortal1.




Post5
Website* host
Person* author
vector<Person*> contributors
vector<Person*> views
Post6
Website* host
Person* author
vector<Person*> contributors
vector<Person*> views
Post7
Website* host
Person* author
vector<Person*> contributors
vector<Person*> views
Post8
Website* host
Person* author
vector<Person*> contributors
vector<Person*> views
Person7
Person8
Person9
Person10
Person11
Person12
User7
Website* site
User8
Website* site
User9
Website* site
User10
Website* site
User11
Website* site
User12
Website* site
Admin7
Website* site
Admin8
Website* site
Admin9
Website* site
Admin10
Website* site
Admin11
Website* site
Admin12
Website* site
Fanpage1
vector<Person*> users
vector<Person*> admins
vector<Post*> posts
vector<Person*> guestVisits
vector<Person*> likes
NewsPortal1
vector<Person*> users
vector<Person*> admins
vector<Post*> posts
Post* mainHeadline
vector<Person*> guestVisits
Fan1
vector<Website*> likedFanpages
Fan2
vector<Website*> likedFanpages
Fan3
vector<Website*> likedFanpages
Fanpage2
Fanpage3

Classes

Inheritance diagram



Declarations

```
class Person {
protected:
	string name;
	string address;
	string IP;
	unsigned int age;
public:
	Person(string name, string address, string IP, unsigned int age);
	virtual ~Person();

	string GetName() const;
	string GetAddress() const;
	string GetIP() const;
	unsigned int GetAge() const;

	// Used to figure out the type of the person when determining permissions
	virtual bool IsGuest() const;
	virtual bool IsUser() const;
	virtual bool IsAdmin() const;

	// Prints the person's data
	friend ostream& operator<<(ostream& output, const Person& person);
};

class Guest : public Person {
protected:
	vector<Website*> siteVisits;
public:
	Guest(string name, string address, string IP, unsigned int age);
	virtual ~Guest();

	void VisitSite(Website* site);
	unsigned int CountSpecificSiteVisits(Website* site) const; // How many times
this person has visited the given website
	unsigned int CountSiteVisits(bool unique = false) const; // How many times
this person has visited anything, or how many unique websites has this person
visited, depending on the flag

	bool IsGuest() const override;

	// Prints the guest's data
	friend ostream& operator<<(ostream& output, const Guest& guest);
};

class Fan : public Guest {
protected:
	vector<Website*> likedFanpages;
public:
	Fan(string name, string address, string IP, unsigned int age);
```

```
        virtual ~Fan();

        bool isFanpageLiked(Fanpage* fp) const;
        bool LikeFanpage(Fanpage* fp);
        bool UnLikeFanpage(Fanpage* fp);

        unsigned int CountLikedFanpages() const;

        // Prints the fan's data
        friend ostream& operator<<(ostream& output, const Fan& fan);
};

class User : public Person {
protected:
        string username;
        string password;
        bool loggedIn;
        Website* site;
public:
        User(string name, string address, string IP, unsigned int age, string
username, string password, Website* site);
        virtual ~User();

        string GetUsername() const;
        string GetPassword() const;
        bool ComparePassword(string password) const; // Checks if the provided
password is correct
        bool IsLoggedIn() const; // Used for permissions
        void SetLoggedIn(bool loggedIn);
        Website* GetSite() const;

        bool IsUser() const override;

        // Prints the user's data
        friend ostream& operator<<(ostream& output, const User& user);
};

class Admin : public User {
public:
        Admin(string name, string address, string IP, unsigned int age, string
username, string password, Website* site);
        virtual ~Admin();

        bool IsAdmin() const override;

        // Prints the admin's data
        friend ostream& operator<<(ostream& output, const Admin& admin);
};

class Website {
protected:
        string description;
        string address;
        vector<Person*> users;
        vector<Person*> admins;
        vector<Person*> guestVisits;
        vector<string> bannedIPs;
        vector<Post*> posts;
        time_t creationTime;

public:
        Website(string description, string address);
        virtual ~Website();

        unsigned int CountUsers() const;
        unsigned int CountAdmins() const;
        unsigned int CountGuestVisits(bool unique = false) const; // Size of
.guestVisits, if unique is true it only counts every individual guest once
```

```
	virtual void RegisterGuestVisit(Guest* guest); // Adds a guest to guestVisits

	User* RegisterUser(Person* person, string username, string password); //
Creates an User from a Person if their IP is not banned
	bool LogIn(string username, string password); // Tries to log-in an user with
these credentials
	bool LogOut(string username); // Logs an user out
	bool RemoveUser(User* user, bool memory = true); // Removes an user from the
site

	virtual Admin* GrantAdminToUser(User* user); // Creats an Admin from an User
if it doesn't exist
	virtual User* DemoteAdmin(Admin* admin, bool memory = true); // Removes an
Admin from the list, then finds and returns the corresponding User

	bool BanIPAddress(string IP); // Adds an IP into banned IPs
	bool UnBanIPAddress(string IP); // Removes an IP from banned IPs if it's
there

	unsigned int RemoveGuestVisitsByGuest(Guest* guest); // Removes all recorded
guest visits from a specific guest
	unsigned int ClearGuestVisits(); // Removes all recorded guest visits
	unsigned int RemoveAllUsers(bool removeAdmins = true);

	bool IsIPBanned(string IP) const;
	bool IsUserRegistered(User* user) const;
	bool IsUserAdmin(User* user) const;
	bool IsAdminRegistered(Admin* admin) const;

	User* FindUserByUsername(string username) const;
	Admin* FindAdminByUsername(string username) const;

	string GetDescription() const;
	string GetAddress() const;
	time_t GetCreationTime() const;

	// Get n-th element of the respective vector
	User* GetUser(unsigned int n) const;
	Admin* GetAdmin(unsigned int n) const;
	Post* GetPost(unsigned int n) const;

	// Checks permissions, which will allow different actions depending on site
type and person type
	virtual int CheckReadPermissions(Person* person) const;
	virtual int CheckWritePermissions(Person* person) const;

	// Sends a request to create a new post on the site if given author has
permissions to do so
	virtual Post* CreatePost(string title, string content, Person* author, bool
requiresLogin);

	// Describes which type of a website it is, used when printing the list of
all websites
	virtual string WebsiteType() const;

	// Prints the website
	friend ostream& operator<<(ostream& output, const Website& website);
};

class Blog : public Website {
private:
	Person* author;

public:
	Blog(string description, string address, Person* author);
	virtual ~Blog();
```

```
        // There is only one admin on a blog
        Admin* GrantAdminToUser(User* user) override;
        User* DemoteAdmin(Admin* admin, bool memory) override;

        int CheckReadPermissions(Person* person) const override;
        int CheckWritePermissions(Person* person) const override;

        string WebsiteType() const override;

        // Prints the blog
        friend ostream& operator<<(ostream& output, const Blog& blog);
};

class CompanyWebsite : public Website {
private:
        string companyName;
        string companyAddress;
        string companyContact;
        Person* lastEditor;

public:
        CompanyWebsite(string description, string address, string companyName, string
companyAddress, string companyContact);
        virtual ~CompanyWebsite();

        int CheckReadPermissions(Person* person) const override;
        int CheckWritePermissions(Person* person) const override;

        // There are no posts on a company website
        Post* CreatePost(string title, string content, Person* author, bool
requiresLogin) override;

        // Updates the company's data (permissive)
        bool UpdateCompanyData(string companyName, string companyAddress, string
companyContact, Person* requestSource);

        string GetCompanyName() const;
        string WebsiteType() const override;

        // Prints the website
        friend ostream& operator<<(ostream& output, const CompanyWebsite&
companyWebsite);
};

class Fanpage : public Website {
private:
        vector<Person*> likes;

public:
        Fanpage(string description, string address);
        virtual ~Fanpage();

        bool DoesFanLikeFanpage(Fan* fan);
        bool RegisterLike(Fan* fan);
        bool RemoveLike(Fan* fan);

        unsigned int CountLikes() const;

        string WebsiteType() const override;

        // Prints the fanpage
        friend ostream& operator<<(ostream& output, const Fanpage& fanpage);
};

class NewsPortal : public Website {
private:
        Post* mainHeadline;

public:
```

```
        NewsPortal(string description, string address);
        virtual ~NewsPortal();

        bool SetMainHeadline(Post* headline, Person* requestSource); // Changes the
headline (permissive)

        void RegisterGuestVisit(Guest* guest) override;

        string WebsiteType() const override;

        friend ostream& operator<<(ostream& output, const NewsPortal& NewsPortal);
};

class Post {
private:
        string title;
        string content;
        Person* author;
        vector<Person*> contributors;
        time_t additionTime;
        time_t lastEditTime;
        Website* host;
        bool requiresLogin;
        vector<Person*> views;

public:
        Post(string title, string content, Person* author, Website* host, bool
requiresLogin);
        ~Post();
        bool EditPost(string newTitle, string newContent, Person* contributor); //
Modify the post if the contributor has permissions to do so
        bool EditVisibility(bool requiresLogin, Person* requestSource); // Change
whether the post is only visible to logged-in users (permissive)

        bool ReadPost(Person* reader); // A person tries to read the post
(permissive)

        unsigned int CountViews(bool unique = true) const;

        Website* GetHost() const;
        Person* GetAuthor() const;
        string GetTitle() const;

        void ClearAuthor(bool clearAdmins = true);
        void ClearContributors(bool clearAdmins = true);
        bool RemoveContributorByUsername(string username);

        bool HasPersonContributed(Person* person) const;
        bool RegisterContribution(Person* person);
        int CountContributors() const;
        Person* GetContributor(unsigned int n) const;

        // Prints the post
        friend ostream& operator<<(ostream& output, const Post& post);
};

class Internet {
private:
        vector<Person*> people;
        vector<Website*> websites;
        string lastIP;

        // Internet is a singleton
        Internet();
        ~Internet();

        string NextIP();
```

```
public:
	Internet(const Internet&);
	Internet operator= (const Internet&);

	static Internet& GetInternet();

	Person* GetPerson(unsigned int n) const;
	Website* GetWebsite(unsigned int n) const;

	Person* FindPersonByName(string name) const;
	Person* FindPersonByIP(string IP) const;
	Website* FindWebsiteByAddress(string address) const;
	CompanyWebsite* FindCompanyWebsiteByCompanyName(string companyName) const;

	Person* CreatePerson(string name, string address, unsigned int age);
	bool RegisterPerson(Person* person);
	bool RegisterWebsite(Website* website);

	bool IsPersonRegistered(Person* person) const;
	bool IsWebsiteRegistered(Website* website) const;

	bool RemovePerson(Person* person);
	bool RemoveWebsite(Website* website);

	void Clear();

	void PrintAllPeople() const;
	void PrintAllWebsites() const;

	// Adds the website/person to the internet
	friend Internet& operator<<(Internet& internet, Website& website);
	friend Internet& operator<<(Internet& internet, Person& person);

	// Removes the website/person from the internet
	friend Internet& operator>>(Internet& internet, Website& website);
	friend Internet& operator>>(Internet& internet, Person& person);


};
```

Testing

```
Person* John = new Person("John", "Test 1/23, Poland", "127.0.0.1", 30);

cout << John->GetAddress() << endl; // Test 1/23, Poland
cout << John->GetName() << endl; // John
cout << John->GetIP() << endl; // 127.0.0.1
cout << John->GetAge() << endl; // 30
cout << John->IsAdmin() << endl; // false
cout << John->IsUser() << endl; // false
cout << John->IsGuest() << endl; // false
cout << (*John) << endl; // All of John's data

Website* TestPage = new Website("An arbitrary website", "testpage.com");

cout << TestPage->WebsiteType() << endl; // "Website"

User* JohnUser = TestPage->RegisterUser(John, "john123", "terriblepassword");

cout << (JohnUser == NULL) << endl; // false

cout << JohnUser->IsUser() << endl; // true
cout << JohnUser->IsAdmin() << endl; // false
cout << JohnUser->IsGuest() << endl; // false
cout << JohnUser->GetUsername() << endl; // john123
```

```
cout << JohnUser->IsLoggedIn() << endl; // false

cout << TestPage->LogIn("john123", "aaaaa123") << endl; // false
cout << JohnUser->IsLoggedIn() << endl; // false
cout << TestPage->LogIn("john123", "terriblepassword") << endl; // true
cout << JohnUser->IsLoggedIn() << endl; // true
cout << TestPage->LogOut("john123") << endl; // true
cout << TestPage->LogOut("john123") << endl; // false
cout << JohnUser->IsLoggedIn() << endl; // false
cout << TestPage->LogIn("john123", "terriblepassword") << endl; // true

Person* Jim = new Person("Jim", "Test 7/13, Poland", "127.0.0.2", 12);
cout << TestPage->BanIPAddress("127.0.0.2") << endl; // true
cout << TestPage->BanIPAddress("127.0.0.2") << endl; // false
User* JimUser = TestPage->RegisterUser(Jim, "jim1", "ngdsg");
cout << (JimUser == NULL) << endl; // true

Guest* randomGuest = new Guest("Irrelevant", "So is this", "127.0.0.3", 30);
for (int i = 0; i < 5; i++) TestPage->RegisterGuestVisit(randomGuest);
cout << TestPage->CountGuestVisits() << endl; // 5
cout << TestPage->CountGuestVisits(true) << endl; // 1

cout << TestPage->IsUserAdmin(JohnUser) << endl; // false
Admin* JohnAdmin = TestPage->GrantAdminToUser(JohnUser);
cout << (JohnAdmin == NULL) << endl; // false
cout << TestPage->CountAdmins() << endl; // 1
cout << TestPage->CountUsers() << endl; // 1

cout << TestPage->IsUserAdmin(JohnUser) << endl; // true
cout << TestPage->IsUserAdmin(JohnAdmin) << endl; // true

cout << TestPage->ClearGuestVisits() << endl; // 5
cout << TestPage->RemoveAllUsers(false) << endl; // 0
cout << TestPage->RemoveAllUsers(true) << endl; // 1

// We know the pointers to JohnUser and JohnAdmin are no longer valid, so we better
set them to null
// This protects us if we forget to keep track
JohnUser = NULL;
JohnAdmin = NULL;

// Printing test
cout << TestPage->GetAddress() << endl; // testpage.com
cout << TestPage->GetDescription() << endl; // An arbitrary website
cout << TestPage->GetCreationTime() << endl; // Should print the system time at the
moment of creation

cout << TestPage->UnBanIPAddress("127.0.0.2") << endl; // true
cout << TestPage->UnBanIPAddress("127.0.0.2") << endl; // false
JimUser = TestPage->RegisterUser(Jim, "jim1", "ngdsg");
cout << (JimUser == NULL) << endl; // false

cout << TestPage->LogIn("jim1", "ngdsg") << endl; // true
cout << TestPage->CheckReadPermissions(JimUser) << endl; // 1
cout << TestPage->CheckWritePermissions(JimUser) << endl; // 1
cout << TestPage->CheckReadPermissions(randomGuest) << endl; // 1
cout << TestPage->CheckWritePermissions(randomGuest) << endl; // 0

Post* TestPost = TestPage->CreatePost("Lorem ipsum", "Dolor sit amet...", JohnUser,
false);
cout << (TestPost == NULL) << endl; // true
TestPost = TestPage->CreatePost("Lorem ipsum", "Dolor sit amet...", JimUser,
false);
cout << (TestPost == NULL) << endl; // false

cout << (*TestPost) << endl; // Should print the post
cout << (*TestPage) << endl; // Should print the website and have it contain the
post
```

```
cout << TestPost->EditPost("Ipsum lorem", "Dolor sit amet.", JimUser) << endl; //
true

cout << (*TestPost) << endl; // Should print the edited post (Ipsum lorem)
cout << (*TestPage) << endl; // Should print the website and have it contain the
edited post

User* JimUserDuplicate = TestPage->RegisterUser(Jim, "jim1", "ngdsg");
cout << (JimUserDuplicate == NULL) << endl; // true

JimUserDuplicate = TestPage->RegisterUser(Jim, "jim2", "ngdsg");
cout << (JimUserDuplicate == NULL) << endl; // false

cout << TestPost->ReadPost(Jim) << endl; // true
cout << TestPost->ReadPost(Jim) << endl; // true
cout << TestPost->ReadPost(Jim) << endl; // true

cout << TestPost->CountViews() << endl; // 3
cout << TestPost->CountViews(true) << endl; // 1

Fan* fan1 = new Fan("Fan1", "Fancity, Fanland", "127.0.0.3", 18);
Fan* fan2 = new Fan("Fan2", "Fancity, Fanland", "127.0.0.3", 18);
Fan* fan3 = new Fan("Fan3", "Fancity, Fanland", "127.0.0.3", 18);

Fanpage* fp1 = new Fanpage("Yet Another Garage Band", "fb.com/yagb");
Fanpage* fp2 = new Fanpage("Daily Memes", "fb.com/lol");
cout << fan1->LikeFanpage(fp1) << endl; // true
cout << fan1->LikeFanpage(fp1) << endl; // false
cout << fan2->LikeFanpage(fp1) << endl; // true
cout << fan3->LikeFanpage(fp1) << endl; // true

cout << fan1->LikeFanpage(fp2) << endl; // true

cout << fan1->CountLikedFanpages() << endl; // 2
cout << fan2->CountLikedFanpages() << endl; // 1
cout << fan3->CountLikedFanpages() << endl; // 1

cout << fp1->CountLikes() << endl; // 3
cout << fp2->CountLikes() << endl; // 1

cout << fan1->UnLikeFanpage(fp2) << endl; // true
cout << fan1->UnLikeFanpage(fp2) << endl; // false

cout << fp2->CountLikes() << endl; // 0

Person* Jinny = new Person("Jinny", "Chicago, Illinois, USA", "127.0.0.4", 19);

Blog* TestBlog = new Blog("Another Fashion Blog", "lookatme.com", Jinny);
cout << TestBlog->CountAdmins() << endl; // 1 (created by default)
Admin* JinnyAdmin = TestBlog->GetAdmin(0);
cout << (JinnyAdmin == NULL) << endl; // false
cout << JinnyAdmin->GetName() << endl; // "Jinny"

User* user1 = TestBlog->RegisterUser(fan1, "abc", "xyz");
cout << (user1 == NULL) << endl; // false

Admin* admin1 = TestBlog->GrantAdminToUser(user1);
cout << (admin1 == NULL) << endl; // true
Admin* admin0 = TestBlog->GrantAdminToUser(NULL);
cout << (admin0 == NULL) << endl; // true

cout << TestBlog->CheckReadPermissions(randomGuest) << endl; // 1
cout << TestBlog->CheckWritePermissions(randomGuest) << endl; // 0
cout << TestBlog->CheckReadPermissions(user1) << endl; // 1
cout << TestBlog->CheckWritePermissions(user1) << endl; // 0

cout << TestBlog->CheckReadPermissions(Jinny) << endl; // 1
```

```
cout << TestBlog->CheckWritePermissions(Jinny) << endl; // 1

Person* Writer1 = new Person("Jack", "irrelevant", "127.0.0.5", 30);
Person* Writer2 = new Person("Zach", "irrelevant", "127.0.0.6", 30);
Person* Writer3 = new Person("Hugh", "irrelevant", "127.0.0.7", 30);

NewsPortal* xyz = new NewsPortal("XYZ News", "xyz.co.uk");
User* WriterUser1 = xyz->RegisterUser(Writer1, "writer1", "pwd1");
User* WriterUser2 = xyz->RegisterUser(Writer2, "writer2", "pwd2");
User* WriterUser3 = xyz->RegisterUser(Writer3, "writer3", "pwd3");
cout << (WriterUser1 == NULL || WriterUser2 == NULL || WriterUser3 == NULL) <<
endl; // false

xyz->LogIn("writer1", "pwd1");
xyz->LogIn("writer2", "pwd2");
xyz->LogIn("writer3", "pwd3");

Admin* WriterAdmin = xyz->GrantAdminToUser(WriterUser1);
cout << (WriterAdmin == NULL) << endl; // false

xyz->CreatePost("Something something the Royal Family", "...", WriterUser1, false);
xyz->CreatePost("Cure for cancer found", "...", WriterUser2, false);
xyz->CreatePost("US scientists prove that 2+2=3.9 for really small 2's", "...",
WriterUser3, false);

Post* NewsPost1 = xyz->GetPost(0);
Post* NewsPost2 = xyz->GetPost(1);
Post* NewsPost3 = xyz->GetPost(2);

xyz->RegisterGuestVisit(randomGuest);
xyz->RegisterGuestVisit(randomGuest);

cout << xyz->SetMainHeadline(NewsPost1, WriterUser2) << endl; // false
cout << xyz->SetMainHeadline(NewsPost1, WriterAdmin) << endl; // true

xyz->RegisterGuestVisit(randomGuest);
xyz->RegisterGuestVisit(randomGuest);
xyz->RegisterGuestVisit(randomGuest);

cout << NewsPost1->CountViews() << endl; // 3
cout << NewsPost2->CountViews() << endl; // 0
cout << NewsPost3->CountViews() << endl; // 0

Person* Mark = new Person("Mark", "Wall Street", "127.0.0.8", 40);
CompanyWebsite* AcquisitionINC = new CompanyWebsite("Acquisition Incorporated -
Integrity, Intelligence", "acqinc.com", "Acquisition Incorporated", "Wall Street",
"1-800-12-34, info@acqinc.com");

cout << (*AcquisitionINC) << endl; // Should present the company's data
User* MarkUser = AcquisitionINC->RegisterUser(Mark, "mark", "xHz%@#4gS");
AcquisitionINC->LogIn("mark", "xHz%@#4gS");

cout << AcquisitionINC->UpdateCompanyData("Acquisition", "Finland", "123-456-789,
info@acqinc.com", MarkUser) << endl; // false

cout << (*AcquisitionINC) << endl; // Should present the company's old data

Admin* MarkAdmin = AcquisitionINC->GrantAdminToUser(MarkUser);

cout << AcquisitionINC->UpdateCompanyData("Acquisition", "Finland", "123-456-789,
info@acqinc.com", MarkAdmin) << endl; // true

cout << (*AcquisitionINC) << endl; // Should present the company's updated data

cout << (AcquisitionINC->CreatePost("a", "b", MarkAdmin, false) == NULL) << endl;
// true

// Put everything into the Internet for easier access
```

```
Internet& net = Internet::GetInternet();

// Should work with single argument
net << *John;
net << *TestPage;

// And with multiple ones
net << *Jim << *randomGuest << *fan1 << *fan2 << *fan3 << *Jinny << *Writer1 <<
*Writer2 << *Writer3;
net << *fp1 << *fp2 << *TestBlog;

// Also of mixed type
net << *xyz << *Mark << *AcquisitionINC;

net.PrintAllPeople(); // Should print all the people in the order they were added
net.PrintAllWebsites(); // Should print all websites (address, description and
type) in the order they were added

net >> *Mark >> *AcquisitionINC;

net.PrintAllPeople(); // Should print all the people sans Mark
net.PrintAllWebsites(); // Should print all websites sans Acq Inc

delete AcquisitionINC;
delete Mark;

net.Clear();
```

Changes

Fan

- Added the "isFanpageLiked" method to check if a fanpage is in the likedFanpages vector already

User

- Added the "GetPassword" and "SetLoggedIn" accessors

Admin

- Admins are now always considered "logged in", provided that their Admin object (rather than the User equivalent) is used for the permissions check

Website

- Added the "RemoveGuestVisitsByGuest" method to remove track of all guest visits made by a specific guest to the given website
- Added the "IsAdminRegistered" method to check if admin is in the admins vector
- Added methods to find users and admins by their usernames
- Changed the argument type of methods that get n-th element of users, admins or posts vectors from int to unsigned int
- The "RemoveUser" and "DemoteAdmin" methods now have an additional bool flag that avoids recursion when it's called from within an user or admin destructor

CompanyWebsite

- Added a "GetCompanyName" accessor

Fanpage

- Added a method to check if a specific fan is in the "likes" vector

Post

- No longer has a private constructor (it's public now) called by a static method - the permission checks handling has been moved to the Website's "CreatePost" method instead
- Added "GetHost", "GetAuthor", "GetTitle", "GetContributor" accessors
- Added methods to set the author pointer or all contributor pointers to NULL (this is used when the users and/or admins are being deleted from a website, to indicate that they no longer exist rather)
- Added a method to remove a specific contributor by username as well
- Added a method to count the number of all contributors
- The original author now counts as a contributor as well and is added to that list

Internet

- Added methods to remove people and websites from the vectors
- Added methods to check if a person or a website is in the respective vector
- Added a method to delete all the listed people and websites
- Changed the singleton pattern implementation from pointers to references