Systemoblig 4

Uke 1

1.

<https://cruise.eecs.uottawa.ca/umpleonline/umple.php?model=190409240337>

2.

<https://cruise.eecs.uottawa.ca/umpleonline/umple.php?model=190409240337>

3.

Står etter oppgave 4

4.

Det som blir generert har flere ulike metoder. Mange av disse metodene ville vi sikkert aldri bruke, men de ser likevel til å være brukelig i noen tilfeller (addVoteAt() og addOrMoveVoteAt()). Andre ting vi har aldri gjørt var boolean wasSet i hver set motode for eksempel. Det blir lagd masse ulike hjelpemetoder som indexOfVote() og numberOfVotes(). Generelt er det bare masse ting for å jobbe rund ArrayList.

3.

Klassediagram:

Kode Java:

//%% NEW FILE Stand BEGINS HERE %%

/\*PLEASE DO NOT EDIT THIS CODE\*/

/\*This code was generated using the UMPLE 1.29.1.4448.81a70243a modeling language!\*/

import java.util.\*;

/\*\*

\* Positioning

\*/

// line 2 "model.ump"

// line 38 "model.ump"

public class Stand

{

//------------------------

// MEMBER VARIABLES

//------------------------

//Stand Attributes

private string nr;

private int totalScore;

//Stand Associations

private List<Vote> votes;

private List<User> users;

//------------------------

// CONSTRUCTOR

//------------------------

public Stand(string aNr, int aTotalScore)

{

nr = aNr;

totalScore = aTotalScore;

votes = new ArrayList<Vote>();

users = new ArrayList<User>();

}

//------------------------

// INTERFACE

//------------------------

public boolean setNr(string aNr)

{

boolean wasSet = false;

nr = aNr;

wasSet = true;

return wasSet;

}

public boolean setTotalScore(int aTotalScore)

{

boolean wasSet = false;

totalScore = aTotalScore;

wasSet = true;

return wasSet;

}

public string getNr()

{

return nr;

}

public int getTotalScore()

{

return totalScore;

}

/\* Code from template association\_GetMany \*/

public Vote getVote(int index)

{

Vote aVote = votes.get(index);

return aVote;

}

public List<Vote> getVotes()

{

List<Vote> newVotes = Collections.unmodifiableList(votes);

return newVotes;

}

public int numberOfVotes()

{

int number = votes.size();

return number;

}

public boolean hasVotes()

{

boolean has = votes.size() > 0;

return has;

}

public int indexOfVote(Vote aVote)

{

int index = votes.indexOf(aVote);

return index;

}

/\* Code from template association\_GetMany \*/

public User getUser(int index)

{

User aUser = users.get(index);

return aUser;

}

public List<User> getUsers()

{

List<User> newUsers = Collections.unmodifiableList(users);

return newUsers;

}

public int numberOfUsers()

{

int number = users.size();

return number;

}

public boolean hasUsers()

{

boolean has = users.size() > 0;

return has;

}

public int indexOfUser(User aUser)

{

int index = users.indexOf(aUser);

return index;

}

/\* Code from template association\_MinimumNumberOfMethod \*/

public static int minimumNumberOfVotes()

{

return 0;

}

/\* Code from template association\_AddManyToOne \*/

public Vote addVote(int aScore, String aTime, User aUser, Statistics aStatistics)

{

return new Vote(aScore, aTime, this, aUser, aStatistics);

}

public boolean addVote(Vote aVote)

{

boolean wasAdded = false;

if (votes.contains(aVote)) { return false; }

Stand existingStand = aVote.getStand();

boolean isNewStand = existingStand != null && !this.equals(existingStand);

if (isNewStand)

{

aVote.setStand(this);

}

else

{

votes.add(aVote);

}

wasAdded = true;

return wasAdded;

}

public boolean removeVote(Vote aVote)

{

boolean wasRemoved = false;

//Unable to remove aVote, as it must always have a stand

if (!this.equals(aVote.getStand()))

{

votes.remove(aVote);

wasRemoved = true;

}

return wasRemoved;

}

/\* Code from template association\_AddIndexControlFunctions \*/

public boolean addVoteAt(Vote aVote, int index)

{

boolean wasAdded = false;

if(addVote(aVote))

{

if(index < 0 ) { index = 0; }

if(index > numberOfVotes()) { index = numberOfVotes() - 1; }

votes.remove(aVote);

votes.add(index, aVote);

wasAdded = true;

}

return wasAdded;

}

public boolean addOrMoveVoteAt(Vote aVote, int index)

{

boolean wasAdded = false;

if(votes.contains(aVote))

{

if(index < 0 ) { index = 0; }

if(index > numberOfVotes()) { index = numberOfVotes() - 1; }

votes.remove(aVote);

votes.add(index, aVote);

wasAdded = true;

}

else

{

wasAdded = addVoteAt(aVote, index);

}

return wasAdded;

}

/\* Code from template association\_MinimumNumberOfMethod \*/

public static int minimumNumberOfUsers()

{

return 0;

}

/\* Code from template association\_AddManyToOne \*/

public User addUser(String aId)

{

return new User(aId, this);

}

public boolean addUser(User aUser)

{

boolean wasAdded = false;

if (users.contains(aUser)) { return false; }

Stand existingStand = aUser.getStand();

boolean isNewStand = existingStand != null && !this.equals(existingStand);

if (isNewStand)

{

aUser.setStand(this);

}

else

{

users.add(aUser);

}

wasAdded = true;

return wasAdded;

}

public boolean removeUser(User aUser)

{

boolean wasRemoved = false;

//Unable to remove aUser, as it must always have a stand

if (!this.equals(aUser.getStand()))

{

users.remove(aUser);

wasRemoved = true;

}

return wasRemoved;

}

/\* Code from template association\_AddIndexControlFunctions \*/

public boolean addUserAt(User aUser, int index)

{

boolean wasAdded = false;

if(addUser(aUser))

{

if(index < 0 ) { index = 0; }

if(index > numberOfUsers()) { index = numberOfUsers() - 1; }

users.remove(aUser);

users.add(index, aUser);

wasAdded = true;

}

return wasAdded;

}

public boolean addOrMoveUserAt(User aUser, int index)

{

boolean wasAdded = false;

if(users.contains(aUser))

{

if(index < 0 ) { index = 0; }

if(index > numberOfUsers()) { index = numberOfUsers() - 1; }

users.remove(aUser);

users.add(index, aUser);

wasAdded = true;

}

else

{

wasAdded = addUserAt(aUser, index);

}

return wasAdded;

}

public void delete()

{

for(int i=votes.size(); i > 0; i--)

{

Vote aVote = votes.get(i - 1);

aVote.delete();

}

for(int i=users.size(); i > 0; i--)

{

User aUser = users.get(i - 1);

aUser.delete();

}

}

public String toString()

{

return super.toString() + "["+

"totalScore" + ":" + getTotalScore()+ "]" + System.getProperties().getProperty("line.separator") +

" " + "nr" + "=" + (getNr() != null ? !getNr().equals(this) ? getNr().toString().replaceAll(" "," ") : "this" : "null");

}

}

//%% NEW FILE Jury BEGINS HERE %%

/\*PLEASE DO NOT EDIT THIS CODE\*/

/\*This code was generated using the UMPLE 1.29.1.4448.81a70243a modeling language!\*/

import java.util.\*;

// line 22 "model.ump"

// line 57 "model.ump"

public class Jury extends User

{

//------------------------

// MEMBER VARIABLES

//------------------------

//Jury Associations

private Statistics statistics;

//------------------------

// CONSTRUCTOR

//------------------------

public Jury(String aId, Stand aStand, Statistics aStatistics)

{

super(aId, aStand);

boolean didAddStatistics = setStatistics(aStatistics);

if (!didAddStatistics)

{

throw new RuntimeException("Unable to create jury due to statistics");

}

}

//------------------------

// INTERFACE

//------------------------

/\* Code from template association\_GetOne \*/

public Statistics getStatistics()

{

return statistics;

}

/\* Code from template association\_SetOneToMany \*/

public boolean setStatistics(Statistics aStatistics)

{

boolean wasSet = false;

if (aStatistics == null)

{

return wasSet;

}

Statistics existingStatistics = statistics;

statistics = aStatistics;

if (existingStatistics != null && !existingStatistics.equals(aStatistics))

{

existingStatistics.removeJury(this);

}

statistics.addJury(this);

wasSet = true;

return wasSet;

}

public void delete()

{

Statistics placeholderStatistics = statistics;

this.statistics = null;

if(placeholderStatistics != null)

{

placeholderStatistics.removeJury(this);

}

super.delete();

}

}

//%% NEW FILE Vote BEGINS HERE %%

/\*PLEASE DO NOT EDIT THIS CODE\*/

/\*This code was generated using the UMPLE 1.29.1.4448.81a70243a modeling language!\*/

// line 10 "model.ump"

// line 46 "model.ump"

public class Vote

{

//------------------------

// MEMBER VARIABLES

//------------------------

//Vote Attributes

private int score;

private String time;

//Vote Associations

private Stand stand;

private User user;

private Statistics statistics;

//------------------------

// CONSTRUCTOR

//------------------------

public Vote(int aScore, String aTime, Stand aStand, User aUser, Statistics aStatistics)

{

score = aScore;

time = aTime;

boolean didAddStand = setStand(aStand);

if (!didAddStand)

{

throw new RuntimeException("Unable to create vote due to stand");

}

boolean didAddUser = setUser(aUser);

if (!didAddUser)

{

throw new RuntimeException("Unable to create vote due to user");

}

boolean didAddStatistics = setStatistics(aStatistics);

if (!didAddStatistics)

{

throw new RuntimeException("Unable to create vote due to statistics");

}

}

//------------------------

// INTERFACE

//------------------------

public boolean setScore(int aScore)

{

boolean wasSet = false;

score = aScore;

wasSet = true;

return wasSet;

}

public boolean setTime(String aTime)

{

boolean wasSet = false;

time = aTime;

wasSet = true;

return wasSet;

}

public int getScore()

{

return score;

}

public String getTime()

{

return time;

}

/\* Code from template association\_GetOne \*/

public Stand getStand()

{

return stand;

}

/\* Code from template association\_GetOne \*/

public User getUser()

{

return user;

}

/\* Code from template association\_GetOne \*/

public Statistics getStatistics()

{

return statistics;

}

/\* Code from template association\_SetOneToMany \*/

public boolean setStand(Stand aStand)

{

boolean wasSet = false;

if (aStand == null)

{

return wasSet;

}

Stand existingStand = stand;

stand = aStand;

if (existingStand != null && !existingStand.equals(aStand))

{

existingStand.removeVote(this);

}

stand.addVote(this);

wasSet = true;

return wasSet;

}

/\* Code from template association\_SetOneToMany \*/

public boolean setUser(User aUser)

{

boolean wasSet = false;

if (aUser == null)

{

return wasSet;

}

User existingUser = user;

user = aUser;

if (existingUser != null && !existingUser.equals(aUser))

{

existingUser.removeVote(this);

}

user.addVote(this);

wasSet = true;

return wasSet;

}

/\* Code from template association\_SetOneToMany \*/

public boolean setStatistics(Statistics aStatistics)

{

boolean wasSet = false;

if (aStatistics == null)

{

return wasSet;

}

Statistics existingStatistics = statistics;

statistics = aStatistics;

if (existingStatistics != null && !existingStatistics.equals(aStatistics))

{

existingStatistics.removeVote(this);

}

statistics.addVote(this);

wasSet = true;

return wasSet;

}

public void delete()

{

Stand placeholderStand = stand;

this.stand = null;

if(placeholderStand != null)

{

placeholderStand.removeVote(this);

}

User placeholderUser = user;

this.user = null;

if(placeholderUser != null)

{

placeholderUser.removeVote(this);

}

Statistics placeholderStatistics = statistics;

this.statistics = null;

if(placeholderStatistics != null)

{

placeholderStatistics.removeVote(this);

}

}

public String toString()

{

return super.toString() + "["+

"score" + ":" + getScore()+ "," +

"time" + ":" + getTime()+ "]" + System.getProperties().getProperty("line.separator") +

" " + "stand = "+(getStand()!=null?Integer.toHexString(System.identityHashCode(getStand())):"null") + System.getProperties().getProperty("line.separator") +

" " + "user = "+(getUser()!=null?Integer.toHexString(System.identityHashCode(getUser())):"null") + System.getProperties().getProperty("line.separator") +

" " + "statistics = "+(getStatistics()!=null?Integer.toHexString(System.identityHashCode(getStatistics())):"null");

}

}

//%% NEW FILE User BEGINS HERE %%

/\*PLEASE DO NOT EDIT THIS CODE\*/

/\*This code was generated using the UMPLE 1.29.1.4448.81a70243a modeling language!\*/

import java.util.\*;

// line 16 "model.ump"

// line 51 "model.ump"

public class User

{

//------------------------

// MEMBER VARIABLES

//------------------------

//User Attributes

private String id;

//User Associations

private List<Vote> votes;

private Stand stand;

//------------------------

// CONSTRUCTOR

//------------------------

public User(String aId, Stand aStand)

{

id = aId;

votes = new ArrayList<Vote>();

boolean didAddStand = setStand(aStand);

if (!didAddStand)

{

throw new RuntimeException("Unable to create user due to stand");

}

}

//------------------------

// INTERFACE

//------------------------

public boolean setId(String aId)

{

boolean wasSet = false;

id = aId;

wasSet = true;

return wasSet;

}

public String getId()

{

return id;

}

/\* Code from template association\_GetMany \*/

public Vote getVote(int index)

{

Vote aVote = votes.get(index);

return aVote;

}

public List<Vote> getVotes()

{

List<Vote> newVotes = Collections.unmodifiableList(votes);

return newVotes;

}

public int numberOfVotes()

{

int number = votes.size();

return number;

}

public boolean hasVotes()

{

boolean has = votes.size() > 0;

return has;

}

public int indexOfVote(Vote aVote)

{

int index = votes.indexOf(aVote);

return index;

}

/\* Code from template association\_GetOne \*/

public Stand getStand()

{

return stand;

}

/\* Code from template association\_MinimumNumberOfMethod \*/

public static int minimumNumberOfVotes()

{

return 0;

}

/\* Code from template association\_AddManyToOne \*/

public Vote addVote(int aScore, String aTime, Stand aStand, Statistics aStatistics)

{

return new Vote(aScore, aTime, aStand, this, aStatistics);

}

public boolean addVote(Vote aVote)

{

boolean wasAdded = false;

if (votes.contains(aVote)) { return false; }

User existingUser = aVote.getUser();

boolean isNewUser = existingUser != null && !this.equals(existingUser);

if (isNewUser)

{

aVote.setUser(this);

}

else

{

votes.add(aVote);

}

wasAdded = true;

return wasAdded;

}

public boolean removeVote(Vote aVote)

{

boolean wasRemoved = false;

//Unable to remove aVote, as it must always have a user

if (!this.equals(aVote.getUser()))

{

votes.remove(aVote);

wasRemoved = true;

}

return wasRemoved;

}

/\* Code from template association\_AddIndexControlFunctions \*/

public boolean addVoteAt(Vote aVote, int index)

{

boolean wasAdded = false;

if(addVote(aVote))

{

if(index < 0 ) { index = 0; }

if(index > numberOfVotes()) { index = numberOfVotes() - 1; }

votes.remove(aVote);

votes.add(index, aVote);

wasAdded = true;

}

return wasAdded;

}

public boolean addOrMoveVoteAt(Vote aVote, int index)

{

boolean wasAdded = false;

if(votes.contains(aVote))

{

if(index < 0 ) { index = 0; }

if(index > numberOfVotes()) { index = numberOfVotes() - 1; }

votes.remove(aVote);

votes.add(index, aVote);

wasAdded = true;

}

else

{

wasAdded = addVoteAt(aVote, index);

}

return wasAdded;

}

/\* Code from template association\_SetOneToMany \*/

public boolean setStand(Stand aStand)

{

boolean wasSet = false;

if (aStand == null)

{

return wasSet;

}

Stand existingStand = stand;

stand = aStand;

if (existingStand != null && !existingStand.equals(aStand))

{

existingStand.removeUser(this);

}

stand.addUser(this);

wasSet = true;

return wasSet;

}

public void delete()

{

for(int i=votes.size(); i > 0; i--)

{

Vote aVote = votes.get(i - 1);

aVote.delete();

}

Stand placeholderStand = stand;

this.stand = null;

if(placeholderStand != null)

{

placeholderStand.removeUser(this);

}

}

public String toString()

{

return super.toString() + "["+

"id" + ":" + getId()+ "]" + System.getProperties().getProperty("line.separator") +

" " + "stand = "+(getStand()!=null?Integer.toHexString(System.identityHashCode(getStand())):"null");

}

}

//%% NEW FILE Statistics BEGINS HERE %%

/\*PLEASE DO NOT EDIT THIS CODE\*/

/\*This code was generated using the UMPLE 1.29.1.4448.81a70243a modeling language!\*/

import java.util.\*;

// line 28 "model.ump"

// line 64 "model.ump"

public class Statistics

{

//------------------------

// MEMBER VARIABLES

//------------------------

//Statistics Attributes

private String password;

//Statistics Associations

private List<Vote> votes;

private List<Jury> juries;

//------------------------

// CONSTRUCTOR

//------------------------

public Statistics(String aPassword)

{

password = aPassword;

votes = new ArrayList<Vote>();

juries = new ArrayList<Jury>();

}

//------------------------

// INTERFACE

//------------------------

public boolean setPassword(String aPassword)

{

boolean wasSet = false;

password = aPassword;

wasSet = true;

return wasSet;

}

public String getPassword()

{

return password;

}

/\* Code from template association\_GetMany \*/

public Vote getVote(int index)

{

Vote aVote = votes.get(index);

return aVote;

}

public List<Vote> getVotes()

{

List<Vote> newVotes = Collections.unmodifiableList(votes);

return newVotes;

}

public int numberOfVotes()

{

int number = votes.size();

return number;

}

public boolean hasVotes()

{

boolean has = votes.size() > 0;

return has;

}

public int indexOfVote(Vote aVote)

{

int index = votes.indexOf(aVote);

return index;

}

/\* Code from template association\_GetMany \*/

public Jury getJury(int index)

{

Jury aJury = juries.get(index);

return aJury;

}

public List<Jury> getJuries()

{

List<Jury> newJuries = Collections.unmodifiableList(juries);

return newJuries;

}

public int numberOfJuries()

{

int number = juries.size();

return number;

}

public boolean hasJuries()

{

boolean has = juries.size() > 0;

return has;

}

public int indexOfJury(Jury aJury)

{

int index = juries.indexOf(aJury);

return index;

}

/\* Code from template association\_MinimumNumberOfMethod \*/

public static int minimumNumberOfVotes()

{

return 0;

}

/\* Code from template association\_AddManyToOne \*/

public Vote addVote(int aScore, String aTime, Stand aStand, User aUser)

{

return new Vote(aScore, aTime, aStand, aUser, this);

}

public boolean addVote(Vote aVote)

{

boolean wasAdded = false;

if (votes.contains(aVote)) { return false; }

Statistics existingStatistics = aVote.getStatistics();

boolean isNewStatistics = existingStatistics != null && !this.equals(existingStatistics);

if (isNewStatistics)

{

aVote.setStatistics(this);

}

else

{

votes.add(aVote);

}

wasAdded = true;

return wasAdded;

}

public boolean removeVote(Vote aVote)

{

boolean wasRemoved = false;

//Unable to remove aVote, as it must always have a statistics

if (!this.equals(aVote.getStatistics()))

{

votes.remove(aVote);

wasRemoved = true;

}

return wasRemoved;

}

/\* Code from template association\_AddIndexControlFunctions \*/

public boolean addVoteAt(Vote aVote, int index)

{

boolean wasAdded = false;

if(addVote(aVote))

{

if(index < 0 ) { index = 0; }

if(index > numberOfVotes()) { index = numberOfVotes() - 1; }

votes.remove(aVote);

votes.add(index, aVote);

wasAdded = true;

}

return wasAdded;

}

public boolean addOrMoveVoteAt(Vote aVote, int index)

{

boolean wasAdded = false;

if(votes.contains(aVote))

{

if(index < 0 ) { index = 0; }

if(index > numberOfVotes()) { index = numberOfVotes() - 1; }

votes.remove(aVote);

votes.add(index, aVote);

wasAdded = true;

}

else

{

wasAdded = addVoteAt(aVote, index);

}

return wasAdded;

}

/\* Code from template association\_MinimumNumberOfMethod \*/

public static int minimumNumberOfJuries()

{

return 0;

}

/\* Code from template association\_AddManyToOne \*/

public Jury addJury(String aId, Stand aStand)

{

return new Jury(aId, aStand, this);

}

public boolean addJury(Jury aJury)

{

boolean wasAdded = false;

if (juries.contains(aJury)) { return false; }

Statistics existingStatistics = aJury.getStatistics();

boolean isNewStatistics = existingStatistics != null && !this.equals(existingStatistics);

if (isNewStatistics)

{

aJury.setStatistics(this);

}

else

{

juries.add(aJury);

}

wasAdded = true;

return wasAdded;

}

public boolean removeJury(Jury aJury)

{

boolean wasRemoved = false;

//Unable to remove aJury, as it must always have a statistics

if (!this.equals(aJury.getStatistics()))

{

juries.remove(aJury);

wasRemoved = true;

}

return wasRemoved;

}

/\* Code from template association\_AddIndexControlFunctions \*/

public boolean addJuryAt(Jury aJury, int index)

{

boolean wasAdded = false;

if(addJury(aJury))

{

if(index < 0 ) { index = 0; }

if(index > numberOfJuries()) { index = numberOfJuries() - 1; }

juries.remove(aJury);

juries.add(index, aJury);

wasAdded = true;

}

return wasAdded;

}

public boolean addOrMoveJuryAt(Jury aJury, int index)

{

boolean wasAdded = false;

if(juries.contains(aJury))

{

if(index < 0 ) { index = 0; }

if(index > numberOfJuries()) { index = numberOfJuries() - 1; }

juries.remove(aJury);

juries.add(index, aJury);

wasAdded = true;

}

else

{

wasAdded = addJuryAt(aJury, index);

}

return wasAdded;

}

public void delete()

{

for(int i=votes.size(); i > 0; i--)

{

Vote aVote = votes.get(i - 1);

aVote.delete();

}

for(int i=juries.size(); i > 0; i--)

{

Jury aJury = juries.get(i - 1);

aJury.delete();

}

}

public String toString()

{

return super.toString() + "["+

"password" + ":" + getPassword()+ "]";

}

}

Kode PHP:

//%% NEW FILE Stand BEGINS HERE %%

/\*PLEASE DO NOT EDIT THIS CODE\*/

/\*This code was generated using the UMPLE 1.29.1.4448.81a70243a modeling language!\*/

// Positioning

class Stand

{

//------------------------

// MEMBER VARIABLES

//------------------------

//Stand Attributes

private $nr;

private $totalScore;

//Stand Associations

private $votes;

private $users;

//------------------------

// CONSTRUCTOR

//------------------------

public function \_\_construct($aNr, $aTotalScore)

{

$this->nr = $aNr;

$this->totalScore = $aTotalScore;

$this->votes = array();

$this->users = array();

}

//------------------------

// INTERFACE

//------------------------

public function setNr($aNr)

{

$wasSet = false;

$this->nr = $aNr;

$wasSet = true;

return $wasSet;

}

public function setTotalScore($aTotalScore)

{

$wasSet = false;

$this->totalScore = $aTotalScore;

$wasSet = true;

return $wasSet;

}

public function getNr()

{

return $this->nr;

}

public function getTotalScore()

{

return $this->totalScore;

}

public function getVote\_index($index)

{

$aVote = $this->votes[$index];

return $aVote;

}

public function getVotes()

{

$newVotes = $this->votes;

return $newVotes;

}

public function numberOfVotes()

{

$number = count($this->votes);

return $number;

}

public function hasVotes()

{

$has = $this->numberOfVotes() > 0;

return $has;

}

public function indexOfVote($aVote)

{

$wasFound = false;

$index = 0;

foreach($this->votes as $vote)

{

if ($vote->equals($aVote))

{

$wasFound = true;

break;

}

$index += 1;

}

$index = $wasFound ? $index : -1;

return $index;

}

public function getUser\_index($index)

{

$aUser = $this->users[$index];

return $aUser;

}

public function getUsers()

{

$newUsers = $this->users;

return $newUsers;

}

public function numberOfUsers()

{

$number = count($this->users);

return $number;

}

public function hasUsers()

{

$has = $this->numberOfUsers() > 0;

return $has;

}

public function indexOfUser($aUser)

{

$wasFound = false;

$index = 0;

foreach($this->users as $user)

{

if ($user->equals($aUser))

{

$wasFound = true;

break;

}

$index += 1;

}

$index = $wasFound ? $index : -1;

return $index;

}

public static function minimumNumberOfVotes()

{

return 0;

}

public function addVoteVia($aScore, $aTime, $aUser, $aStatistics)

{

return new Vote($aScore, $aTime, $this, $aUser, $aStatistics);

}

public function addVote($aVote)

{

$wasAdded = false;

if ($this->indexOfVote($aVote) !== -1) { return false; }

$existingStand = $aVote->getStand();

$isNewStand = $existingStand != null && $this !== $existingStand;

if ($isNewStand)

{

$aVote->setStand($this);

}

else

{

$this->votes[] = $aVote;

}

$wasAdded = true;

return $wasAdded;

}

public function removeVote($aVote)

{

$wasRemoved = false;

//Unable to remove aVote, as it must always have a stand

if ($this !== $aVote->getStand())

{

unset($this->votes[$this->indexOfVote($aVote)]);

$this->votes = array\_values($this->votes);

$wasRemoved = true;

}

return $wasRemoved;

}

public function addVoteAt($aVote, $index)

{

$wasAdded = false;

if($this->addVote($aVote))

{

if($index < 0 ) { $index = 0; }

if($index > $this->numberOfVotes()) { $index = $this->numberOfVotes() - 1; }

array\_splice($this->votes, $this->indexOfVote($aVote), 1);

array\_splice($this->votes, $index, 0, array($aVote));

$wasAdded = true;

}

return $wasAdded;

}

public function addOrMoveVoteAt($aVote, $index)

{

$wasAdded = false;

if($this->indexOfVote($aVote) !== -1)

{

if($index < 0 ) { $index = 0; }

if($index > $this->numberOfVotes()) { $index = $this->numberOfVotes() - 1; }

array\_splice($this->votes, $this->indexOfVote($aVote), 1);

array\_splice($this->votes, $index, 0, array($aVote));

$wasAdded = true;

}

else

{

$wasAdded = $this->addVoteAt($aVote, $index);

}

return $wasAdded;

}

public static function minimumNumberOfUsers()

{

return 0;

}

public function addUserVia($aId)

{

return new User($aId, $this);

}

public function addUser($aUser)

{

$wasAdded = false;

if ($this->indexOfUser($aUser) !== -1) { return false; }

$existingStand = $aUser->getStand();

$isNewStand = $existingStand != null && $this !== $existingStand;

if ($isNewStand)

{

$aUser->setStand($this);

}

else

{

$this->users[] = $aUser;

}

$wasAdded = true;

return $wasAdded;

}

public function removeUser($aUser)

{

$wasRemoved = false;

//Unable to remove aUser, as it must always have a stand

if ($this !== $aUser->getStand())

{

unset($this->users[$this->indexOfUser($aUser)]);

$this->users = array\_values($this->users);

$wasRemoved = true;

}

return $wasRemoved;

}

public function addUserAt($aUser, $index)

{

$wasAdded = false;

if($this->addUser($aUser))

{

if($index < 0 ) { $index = 0; }

if($index > $this->numberOfUsers()) { $index = $this->numberOfUsers() - 1; }

array\_splice($this->users, $this->indexOfUser($aUser), 1);

array\_splice($this->users, $index, 0, array($aUser));

$wasAdded = true;

}

return $wasAdded;

}

public function addOrMoveUserAt($aUser, $index)

{

$wasAdded = false;

if($this->indexOfUser($aUser) !== -1)

{

if($index < 0 ) { $index = 0; }

if($index > $this->numberOfUsers()) { $index = $this->numberOfUsers() - 1; }

array\_splice($this->users, $this->indexOfUser($aUser), 1);

array\_splice($this->users, $index, 0, array($aUser));

$wasAdded = true;

}

else

{

$wasAdded = $this->addUserAt($aUser, $index);

}

return $wasAdded;

}

public function equals($compareTo)

{

return $this == $compareTo;

}

public function delete()

{

foreach ($this->votes as $aVote)

{

$aVote->delete();

}

foreach ($this->users as $aUser)

{

$aUser->delete();

}

}

}

//%% NEW FILE Jury BEGINS HERE %%

/\*PLEASE DO NOT EDIT THIS CODE\*/

/\*This code was generated using the UMPLE 1.29.1.4448.81a70243a modeling language!\*/

class Jury extends User

{

//------------------------

// MEMBER VARIABLES

//------------------------

//Jury Associations

private $statistics;

//------------------------

// CONSTRUCTOR

//------------------------

public function \_\_construct($aId, $aStand, $aStatistics)

{

parent::\_\_construct($aId, $aStand);

$didAddStatistics = $this->setStatistics($aStatistics);

if (!$didAddStatistics)

{

throw new Exception("Unable to create jury due to statistics");

}

}

//------------------------

// INTERFACE

//------------------------

public function getStatistics()

{

return $this->statistics;

}

public function setStatistics($aStatistics)

{

$wasSet = false;

if ($aStatistics == null)

{

return $wasSet;

}

$existingStatistics = $this->statistics;

$this->statistics = $aStatistics;

if ($existingStatistics != null && $existingStatistics != $aStatistics)

{

$existingStatistics->removeJury($this);

}

$this->statistics->addJury($this);

$wasSet = true;

return $wasSet;

}

public function equals($compareTo)

{

return $this == $compareTo;

}

public function delete()

{

$placeholderStatistics = $this->statistics;

$this->statistics = null;

$placeholderStatistics->removeJury($this);

parent::delete();

}

}

//%% NEW FILE Vote BEGINS HERE %%

/\*PLEASE DO NOT EDIT THIS CODE\*/

/\*This code was generated using the UMPLE 1.29.1.4448.81a70243a modeling language!\*/

class Vote

{

//------------------------

// MEMBER VARIABLES

//------------------------

//Vote Attributes

private $score;

private $time;

//Vote Associations

private $stand;

private $user;

private $statistics;

//------------------------

// CONSTRUCTOR

//------------------------

public function \_\_construct($aScore, $aTime, $aStand, $aUser, $aStatistics)

{

$this->score = $aScore;

$this->time = $aTime;

$didAddStand = $this->setStand($aStand);

if (!$didAddStand)

{

throw new Exception("Unable to create vote due to stand");

}

$didAddUser = $this->setUser($aUser);

if (!$didAddUser)

{

throw new Exception("Unable to create vote due to user");

}

$didAddStatistics = $this->setStatistics($aStatistics);

if (!$didAddStatistics)

{

throw new Exception("Unable to create vote due to statistics");

}

}

//------------------------

// INTERFACE

//------------------------

public function setScore($aScore)

{

$wasSet = false;

$this->score = $aScore;

$wasSet = true;

return $wasSet;

}

public function setTime($aTime)

{

$wasSet = false;

$this->time = $aTime;

$wasSet = true;

return $wasSet;

}

public function getScore()

{

return $this->score;

}

public function getTime()

{

return $this->time;

}

public function getStand()

{

return $this->stand;

}

public function getUser()

{

return $this->user;

}

public function getStatistics()

{

return $this->statistics;

}

public function setStand($aStand)

{

$wasSet = false;

if ($aStand == null)

{

return $wasSet;

}

$existingStand = $this->stand;

$this->stand = $aStand;

if ($existingStand != null && $existingStand != $aStand)

{

$existingStand->removeVote($this);

}

$this->stand->addVote($this);

$wasSet = true;

return $wasSet;

}

public function setUser($aUser)

{

$wasSet = false;

if ($aUser == null)

{

return $wasSet;

}

$existingUser = $this->user;

$this->user = $aUser;

if ($existingUser != null && $existingUser != $aUser)

{

$existingUser->removeVote($this);

}

$this->user->addVote($this);

$wasSet = true;

return $wasSet;

}

public function setStatistics($aStatistics)

{

$wasSet = false;

if ($aStatistics == null)

{

return $wasSet;

}

$existingStatistics = $this->statistics;

$this->statistics = $aStatistics;

if ($existingStatistics != null && $existingStatistics != $aStatistics)

{

$existingStatistics->removeVote($this);

}

$this->statistics->addVote($this);

$wasSet = true;

return $wasSet;

}

public function equals($compareTo)

{

return $this == $compareTo;

}

public function delete()

{

$placeholderStand = $this->stand;

$this->stand = null;

$placeholderStand->removeVote($this);

$placeholderUser = $this->user;

$this->user = null;

$placeholderUser->removeVote($this);

$placeholderStatistics = $this->statistics;

$this->statistics = null;

$placeholderStatistics->removeVote($this);

}

}

//%% NEW FILE User BEGINS HERE %%

/\*PLEASE DO NOT EDIT THIS CODE\*/

/\*This code was generated using the UMPLE 1.29.1.4448.81a70243a modeling language!\*/

class User

{

//------------------------

// MEMBER VARIABLES

//------------------------

//User Attributes

private $id;

//User Associations

private $votes;

private $stand;

//------------------------

// CONSTRUCTOR

//------------------------

public function \_\_construct($aId, $aStand)

{

$this->id = $aId;

$this->votes = array();

$didAddStand = $this->setStand($aStand);

if (!$didAddStand)

{

throw new Exception("Unable to create user due to stand");

}

}

//------------------------

// INTERFACE

//------------------------

public function setId($aId)

{

$wasSet = false;

$this->id = $aId;

$wasSet = true;

return $wasSet;

}

public function getId()

{

return $this->id;

}

public function getVote\_index($index)

{

$aVote = $this->votes[$index];

return $aVote;

}

public function getVotes()

{

$newVotes = $this->votes;

return $newVotes;

}

public function numberOfVotes()

{

$number = count($this->votes);

return $number;

}

public function hasVotes()

{

$has = $this->numberOfVotes() > 0;

return $has;

}

public function indexOfVote($aVote)

{

$wasFound = false;

$index = 0;

foreach($this->votes as $vote)

{

if ($vote->equals($aVote))

{

$wasFound = true;

break;

}

$index += 1;

}

$index = $wasFound ? $index : -1;

return $index;

}

public function getStand()

{

return $this->stand;

}

public static function minimumNumberOfVotes()

{

return 0;

}

public function addVoteVia($aScore, $aTime, $aStand, $aStatistics)

{

return new Vote($aScore, $aTime, $aStand, $this, $aStatistics);

}

public function addVote($aVote)

{

$wasAdded = false;

if ($this->indexOfVote($aVote) !== -1) { return false; }

if ($this->indexOfVote($aVote) !== -1) { return false; }

$existingUser = $aVote->getUser();

$isNewUser = $existingUser != null && $this !== $existingUser;

if ($isNewUser)

{

$aVote->setUser($this);

}

else

{

$this->votes[] = $aVote;

}

$wasAdded = true;

return $wasAdded;

}

public function removeVote($aVote)

{

$wasRemoved = false;

//Unable to remove aVote, as it must always have a user

if ($this !== $aVote->getUser())

{

unset($this->votes[$this->indexOfVote($aVote)]);

$this->votes = array\_values($this->votes);

$wasRemoved = true;

}

return $wasRemoved;

}

public function addVoteAt($aVote, $index)

{

$wasAdded = false;

if($this->addVote($aVote))

{

if($index < 0 ) { $index = 0; }

if($index > $this->numberOfVotes()) { $index = $this->numberOfVotes() - 1; }

array\_splice($this->votes, $this->indexOfVote($aVote), 1);

array\_splice($this->votes, $index, 0, array($aVote));

$wasAdded = true;

}

return $wasAdded;

}

public function addOrMoveVoteAt($aVote, $index)

{

$wasAdded = false;

if($this->indexOfVote($aVote) !== -1)

{

if($index < 0 ) { $index = 0; }

if($index > $this->numberOfVotes()) { $index = $this->numberOfVotes() - 1; }

array\_splice($this->votes, $this->indexOfVote($aVote), 1);

array\_splice($this->votes, $index, 0, array($aVote));

$wasAdded = true;

}

else

{

$wasAdded = $this->addVoteAt($aVote, $index);

}

return $wasAdded;

}

public function setStand($aStand)

{

$wasSet = false;

if ($aStand == null)

{

return $wasSet;

}

$existingStand = $this->stand;

$this->stand = $aStand;

if ($existingStand != null && $existingStand != $aStand)

{

$existingStand->removeUser($this);

}

$this->stand->addUser($this);

$wasSet = true;

return $wasSet;

}

public function equals($compareTo)

{

return $this == $compareTo;

}

public function delete()

{

foreach ($this->votes as $aVote)

{

$aVote->delete();

}

$placeholderStand = $this->stand;

$this->stand = null;

$placeholderStand->removeUser($this);

}

}

//%% NEW FILE Statistics BEGINS HERE %%

/\*PLEASE DO NOT EDIT THIS CODE\*/

/\*This code was generated using the UMPLE 1.29.1.4448.81a70243a modeling language!\*/

class Statistics

{

//------------------------

// MEMBER VARIABLES

//------------------------

//Statistics Attributes

private $password;

//Statistics Associations

private $votes;

private $juries;

//------------------------

// CONSTRUCTOR

//------------------------

public function \_\_construct($aPassword)

{

$this->password = $aPassword;

$this->votes = array();

$this->juries = array();

}

//------------------------

// INTERFACE

//------------------------

public function setPassword($aPassword)

{

$wasSet = false;

$this->password = $aPassword;

$wasSet = true;

return $wasSet;

}

public function getPassword()

{

return $this->password;

}

public function getVote\_index($index)

{

$aVote = $this->votes[$index];

return $aVote;

}

public function getVotes()

{

$newVotes = $this->votes;

return $newVotes;

}

public function numberOfVotes()

{

$number = count($this->votes);

return $number;

}

public function hasVotes()

{

$has = $this->numberOfVotes() > 0;

return $has;

}

public function indexOfVote($aVote)

{

$wasFound = false;

$index = 0;

foreach($this->votes as $vote)

{

if ($vote->equals($aVote))

{

$wasFound = true;

break;

}

$index += 1;

}

$index = $wasFound ? $index : -1;

return $index;

}

public function getJury\_index($index)

{

$aJury = $this->juries[$index];

return $aJury;

}

public function getJuries()

{

$newJuries = $this->juries;

return $newJuries;

}

public function numberOfJuries()

{

$number = count($this->juries);

return $number;

}

public function hasJuries()

{

$has = $this->numberOfJuries() > 0;

return $has;

}

public function indexOfJury($aJury)

{

$wasFound = false;

$index = 0;

foreach($this->juries as $jury)

{

if ($jury->equals($aJury))

{

$wasFound = true;

break;

}

$index += 1;

}

$index = $wasFound ? $index : -1;

return $index;

}

public static function minimumNumberOfVotes()

{

return 0;

}

public function addVoteVia($aScore, $aTime, $aStand, $aUser)

{

return new Vote($aScore, $aTime, $aStand, $aUser, $this);

}

public function addVote($aVote)

{

$wasAdded = false;

if ($this->indexOfVote($aVote) !== -1) { return false; }

$existingStatistics = $aVote->getStatistics();

$isNewStatistics = $existingStatistics != null && $this !== $existingStatistics;

if ($isNewStatistics)

{

$aVote->setStatistics($this);

}

else

{

$this->votes[] = $aVote;

}

$wasAdded = true;

return $wasAdded;

}

public function removeVote($aVote)

{

$wasRemoved = false;

//Unable to remove aVote, as it must always have a statistics

if ($this !== $aVote->getStatistics())

{

unset($this->votes[$this->indexOfVote($aVote)]);

$this->votes = array\_values($this->votes);

$wasRemoved = true;

}

return $wasRemoved;

}

public function addVoteAt($aVote, $index)

{

$wasAdded = false;

if($this->addVote($aVote))

{

if($index < 0 ) { $index = 0; }

if($index > $this->numberOfVotes()) { $index = $this->numberOfVotes() - 1; }

array\_splice($this->votes, $this->indexOfVote($aVote), 1);

array\_splice($this->votes, $index, 0, array($aVote));

$wasAdded = true;

}

return $wasAdded;

}

public function addOrMoveVoteAt($aVote, $index)

{

$wasAdded = false;

if($this->indexOfVote($aVote) !== -1)

{

if($index < 0 ) { $index = 0; }

if($index > $this->numberOfVotes()) { $index = $this->numberOfVotes() - 1; }

array\_splice($this->votes, $this->indexOfVote($aVote), 1);

array\_splice($this->votes, $index, 0, array($aVote));

$wasAdded = true;

}

else

{

$wasAdded = $this->addVoteAt($aVote, $index);

}

return $wasAdded;

}

public static function minimumNumberOfJuries()

{

return 0;

}

public function addJuryVia($aId, $aStand)

{

return new Jury($aId, $aStand, $this);

}

public function addJury($aJury)

{

$wasAdded = false;

if ($this->indexOfJury($aJury) !== -1) { return false; }

$existingStatistics = $aJury->getStatistics();

$isNewStatistics = $existingStatistics != null && $this !== $existingStatistics;

if ($isNewStatistics)

{

$aJury->setStatistics($this);

}

else

{

$this->juries[] = $aJury;

}

$wasAdded = true;

return $wasAdded;

}

public function removeJury($aJury)

{

$wasRemoved = false;

//Unable to remove aJury, as it must always have a statistics

if ($this !== $aJury->getStatistics())

{

unset($this->juries[$this->indexOfJury($aJury)]);

$this->juries = array\_values($this->juries);

$wasRemoved = true;

}

return $wasRemoved;

}

public function addJuryAt($aJury, $index)

{

$wasAdded = false;

if($this->addJury($aJury))

{

if($index < 0 ) { $index = 0; }

if($index > $this->numberOfJuries()) { $index = $this->numberOfJuries() - 1; }

array\_splice($this->juries, $this->indexOfJury($aJury), 1);

array\_splice($this->juries, $index, 0, array($aJury));

$wasAdded = true;

}

return $wasAdded;

}

public function addOrMoveJuryAt($aJury, $index)

{

$wasAdded = false;

if($this->indexOfJury($aJury) !== -1)

{

if($index < 0 ) { $index = 0; }

if($index > $this->numberOfJuries()) { $index = $this->numberOfJuries() - 1; }

array\_splice($this->juries, $this->indexOfJury($aJury), 1);

array\_splice($this->juries, $index, 0, array($aJury));

$wasAdded = true;

}

else

{

$wasAdded = $this->addJuryAt($aJury, $index);

}

return $wasAdded;

}

public function equals($compareTo)

{

return $this == $compareTo;

}

public function delete()

{

foreach ($this->votes as $aVote)

{

$aVote->delete();

}

foreach ($this->juries as $aJury)

{

$aJury->delete();

}

}

}

Kode C++:

//%% NEW FILE Jury.h BEGINS HERE %%

//PLEASE DO NOT EDIT THIS CODE

//This code was generated using the UMPLE 1.29.1.4448.81a70243a modeling language

#ifndef DEF\_\_JURY\_H

#define DEF\_\_JURY\_H

#ifdef PRAGMA

#pragma once

#ifndef \_MSC\_VER

#pragma interface "Jury.h"

#endif

#endif

//------------------------

//FILE INCLUDES

//------------------------

#include <model\_Model.h>

#include <Jury.h>

#include <User.h>

//------------------------

//LIBRARY INCLUDES

//------------------------

using namespace std;

#include <ostream>

class User;

class Statistics;

// line 22 "model.ump"

// line 57 "model.ump"

class Jury : public User{

private:

//Jury Associations

// line 25 "model.ump"

Statistics\* statistics;

public:

//------------------------

//CONSTRUCTOR

//------------------------

Jury(const string aId, Stand\* aStand, Statistics\* aStatistics);

Jury(Jury& other);

//------------------------

//STREAM HELPER GROUPDECLARATION

//------------------------

friend ostream& operator<<(ostream& os, const Jury& dt);

//------------------------

//PREDEFINED OPERATORS

//------------------------

friend bool operator == (Jury& Right, Jury& Left);

friend bool operator != (Jury& Right, Jury& Left){

return !( Right == Left);

}

bool operator == (const Jury& Right) const{

return this == &Right;

}

bool operator != (const Jury& Right) const{

return this != &Right;

}

Jury& operator=(Jury& other);

void internalCopy(Jury& other);

bool setStatistic(Statistics\* aNewStatistic);

inline unsigned int minimumNumberOfStatistics(void){

return 1;

}

// line 0 ""

Statistics\* getStatistics();

// line 0 ""

bool setStatistics(Statistic aStatistics) const;

Statistics\* getStatistic(void);

virtual size\_t hashCode(void);

//------------------------

//DESTRUCTOR

//------------------------

virtual ~Jury();

void deleteAssociatedObjects(void);

protected:

//------------------------

//STREAM HELPER GROUPDECLARATION

//------------------------

virtual void toOstream(ostream& os) const;

};

//------------------------

//GNU HASH FUNCTION USE

//------------------------

#ifdef \_\_GNUC\_\_

using namespace \_\_gnu\_cxx;

namespace \_\_gnu\_cxx{

template<> struct hash<Jury\*>{

size\_t operator()(Jury\* ptr ) const {

return ptr->hashCode();

}

};

}

#include <ext/hash\_map>

#else

#include <hash\_map>

#endif

#endif

//%% NEW FILE model\_Main.cpp BEGINS HERE %%

//PLEASE DO NOT EDIT THIS CODE

//This code was generated using the UMPLE 1.29.1.4448.81a70243a modeling language

#include <Stand.h>

#include <Vote.h>

#include <User.h>

#include <Jury.h>

#include <Statistics.h>

int main(int argc, char \*argv[]){

return 0;

}

//%% NEW FILE model\_Model.h BEGINS HERE %%

//PLEASE DO NOT EDIT THIS CODE

//This code was generated using the UMPLE 1.29.1.4448.81a70243a modeling language

#ifndef DEF\_\_

#define DEF\_\_

#if defined(WIN32) || defined(\_WIN32) || defined(\_\_WIN32\_\_) || defined(\_\_NT\_\_) || defined(\_WIN64)

#define WINDOWS\_OS

// NO PREPROCESSOR DEFINITION FOR PRAGMA

#if \_MSC\_VER

#define PRAGMA

#pragma warning( disable : 4290 )

#endif

#elif defined(hpux) || defined(\_\_hpux) || defined(\_\_hpux\_\_)

#define HPUX\_OS

#elif defined(\_\_APPLE\_\_) || defined(macintosh)

#define MAC\_OS

#elif defined(bsdi) || defined(\_\_bsdi\_\_)

#define BSD\_OS

#endif

#ifdef PRAGMA

#pragma once

#ifdef \_MSC\_VER

#pragma include\_alias("..\\model\_Model.h", "model\_Model.h")

#pragma include\_alias(".\\Stand.h", "/Stand.h")

#pragma include\_alias(".\\Vote.h", "/Vote.h")

#pragma include\_alias(".\\User.h", "/User.h")

#pragma include\_alias(".\\Jury.h", "/Jury.h")

#pragma include\_alias(".\\Statistics.h", "/Statistics.h")

#else

#pragma interface "model\_Model.h"

#endif

#endif

//------------------------

// PACKAGE FILES DECLARATION

//------------------------

#include <sstream>

#include <cmath>

#ifdef WINDOWS\_OS

#include <windows.h>

#include <process.h>

#else

#include <errno.h>

#include <pthread.h>

#include <unistd.h>

#include <cstring>

#include <signal.h>

#endif

#ifdef HPUX\_OS

#include <sys/pstat.h>

#elif defined MAC\_OS

#undef DEBUG

#include <CoreServices/CoreServices.h>

#elif defined BSD\_OS

#include <mach/mach\_types.h>

#include <sys/systm.h>

#include <sys/types.h>

#include <sys/sysctl.h>

#endif

//------------------------

//USED LIBRARIES

//------------------------

using namespace std;

//------------------------

//USED LIBRARIES

//------------------------

#include <vector>

#include <algorithm>

#include <iostream>

#include <cstring>

#include <iostream>

#include <queue>

#include <iostream>

#include <map>

#include <exception>

#include <stdexcept>

#include <cassert>

#include "stdio.h"

//------------------------------

//NAMESPACES AND PREDEFINITIONS

//------------------------------

#ifdef \_\_cplusplus

#endif

//is\_pointer

template <typename T> struct remove\_const\_type { typedef T type; };

template <typename T> struct remove\_const\_type<const T> { typedef T type; };

template <typename T> struct remove\_volatile\_type { typedef T type; };

template <typename T> struct remove\_volatile\_type<volatile T> { typedef T type; };

template <typename T> struct removeType : remove\_const\_type<typename remove\_volatile\_type<T>::type> {};

template <typename T> struct is\_ptr\_type { enum { value = false }; };

template <typename T> struct is\_ptr\_type<T\*> { enum { value = true }; };

template <typename T> struct is\_ptr : is\_ptr\_type<typename removeType<T>::type> {};

#define PLACE\_HOLDER int

#define USECS\_PER\_MSEC 1000

#define MUSECS\_PER\_SEC 1000

#define USECS\_PER\_SEC 1000000

#define INSTANCEOF(object, clazz) !dynamic\_cast<clazz\*>(object)

#define ARGUMENT\_UPPER\_LIMIT 10

#define EMPTY()

#define COMMA() ,

#define SEMICOLON() ;

#define TYPENAME\_ARGS(i, value) typename ArgumentType##i

#define TYPENAME\_VALUE\_ARGS(i, value) typename ArgumentType##i=value

#define INIT\_VALUE\_ARG(i, name) this->\_##name##i=name##i

#define SER\_ARG(i, name) \_##name##i=transport[i - 1]

#define DES\_ARG(i, name) transport[i - 1]=\_##name##i

#define NAMED\_ARG(i, name) name##i

#define MEMBER\_ARG(i, name) ArgumentType##i name##i

#define INIT\_MEMBER\_ARG(i, name) \_##name##i(name##i)

#define VOID\_ARG(i, value) void

#define CAT(a, ...) a ## \_\_VA\_ARGS\_\_

#define REPEAT\_DEC(count ,macro, split, ...) CAT(REPEAT\_DEC\_,count)(macro, split, \_\_VA\_ARGS\_\_)

#define REPEAT\_DEC\_1(macro, split, ...)

#define REPEAT\_DEC\_2(macro, split, ...) macro(1, \_\_VA\_ARGS\_\_)

#define REPEAT\_DEC\_3(macro, split, ...) macro(2, \_\_VA\_ARGS\_\_) split() REPEAT\_DEC\_2(macro, split, \_\_VA\_ARGS\_\_)

#define REPEAT\_DEC\_4(macro, split, ...) macro(3, \_\_VA\_ARGS\_\_) split() REPEAT\_DEC\_3(macro, split, \_\_VA\_ARGS\_\_)

#define REPEAT\_DEC\_5(macro, split, ...) macro(4, \_\_VA\_ARGS\_\_) split() REPEAT\_DEC\_4(macro, split, \_\_VA\_ARGS\_\_)

#define REPEAT\_DEC\_6(macro, split, ...) macro(5, \_\_VA\_ARGS\_\_) split() REPEAT\_DEC\_5(macro, split, \_\_VA\_ARGS\_\_)

#define REPEAT\_DEC\_7(macro, split, ...) macro(6, \_\_VA\_ARGS\_\_) split() REPEAT\_DEC\_6(macro, split, \_\_VA\_ARGS\_\_)

#define REPEAT\_DEC\_8(macro, split, ...) macro(7, \_\_VA\_ARGS\_\_) split() REPEAT\_DEC\_7(macro, split, \_\_VA\_ARGS\_\_)

#define REPEAT\_DEC\_9(macro, split, ...) macro(8, \_\_VA\_ARGS\_\_) split() REPEAT\_DEC\_8(macro, split, \_\_VA\_ARGS\_\_)

#define REPEAT\_DEC\_10(macro, split, ...) macro(9, \_\_VA\_ARGS\_\_) split() REPEAT\_DEC\_9(macro, split, \_\_VA\_ARGS\_\_)

#define REPEAT\_DEC\_11(macro, split, ...) macro(10, \_\_VA\_ARGS\_\_) split() REPEAT\_DEC\_10(macro, split, \_\_VA\_ARGS\_\_)

#define REPEAT\_DEC\_12(macro, split, ...) macro(11, \_\_VA\_ARGS\_\_) split() REPEAT\_DEC\_11(macro, split, \_\_VA\_ARGS\_\_)

#define REPEAT\_DEC\_13(macro, split, ...) macro(12, \_\_VA\_ARGS\_\_) split() REPEAT\_DEC\_12(macro, split, \_\_VA\_ARGS\_\_)

#define REPEAT\_DEC\_14(macro, split, ...) macro(13, \_\_VA\_ARGS\_\_) split() REPEAT\_DEC\_13(macro, split, \_\_VA\_ARGS\_\_)

#define REPEAT\_DEC\_15(macro, split, ...) macro(14, \_\_VA\_ARGS\_\_) split() REPEAT\_DEC\_14(macro, split, \_\_VA\_ARGS\_\_)

#define REPEAT\_DEC\_16(macro, split, ...) macro(15, \_\_VA\_ARGS\_\_) split() REPEAT\_DEC\_15(macro, split, \_\_VA\_ARGS\_\_)

#define REPEAT\_INC(count, macro, split, ...) CAT(REPEAT\_INC\_,count)(macro, split, \_\_VA\_ARGS\_\_)

#define REPEAT\_INC\_1(macro, split, ...) macro(1, \_\_VA\_ARGS\_\_)

#define REPEAT\_INC\_2(macro, split, ...) REPEAT\_INC\_1(macro, split, \_\_VA\_ARGS\_\_) split() macro(2, \_\_VA\_ARGS\_\_)

#define REPEAT\_INC\_3(macro, split, ...) REPEAT\_INC\_2(macro, split, \_\_VA\_ARGS\_\_) split() macro(3, \_\_VA\_ARGS\_\_)

#define REPEAT\_INC\_4(macro, split, ...) REPEAT\_INC\_3(macro, split, \_\_VA\_ARGS\_\_) split() macro(4, \_\_VA\_ARGS\_\_)

#define REPEAT\_INC\_5(macro, split, ...) REPEAT\_INC\_4(macro, split, \_\_VA\_ARGS\_\_) split() macro(5, \_\_VA\_ARGS\_\_)

#define REPEAT\_INC\_6(macro, split, ...) REPEAT\_INC\_5(macro, split, \_\_VA\_ARGS\_\_) split() macro(6, \_\_VA\_ARGS\_\_)

#define REPEAT\_INC\_7(macro, split, ...) REPEAT\_INC\_6(macro, split, \_\_VA\_ARGS\_\_) split() macro(7, \_\_VA\_ARGS\_\_)

#define REPEAT\_INC\_8(macro, split, ...) REPEAT\_INC\_7(macro, split, \_\_VA\_ARGS\_\_) split() macro(8, \_\_VA\_ARGS\_\_)

#define REPEAT\_INC\_9(macro, split, ...) REPEAT\_INC\_8(macro, split, \_\_VA\_ARGS\_\_) split() macro(9, \_\_VA\_ARGS\_\_)

#define REPEAT\_INC\_10(macro, split, ...) REPEAT\_INC\_9(macro, split, \_\_VA\_ARGS\_\_) split() macro(10, \_\_VA\_ARGS\_\_)

#define REPEAT\_INC\_11(macro, split, ...) REPEAT\_INC\_10(macro, split, \_\_VA\_ARGS\_\_) split() macro(11, \_\_VA\_ARGS\_\_)

#define REPEAT\_INC\_12(macro, split, ...) REPEAT\_INC\_11(macro, split, \_\_VA\_ARGS\_\_) split() macro(12, \_\_VA\_ARGS\_\_)

#define REPEAT\_INC\_13(macro, split, ...) REPEAT\_INC\_12(macro, split, \_\_VA\_ARGS\_\_) split() macro(13, \_\_VA\_ARGS\_\_)

#define REPEAT\_INC\_14(macro, split, ...) REPEAT\_INC\_13(macro, split, \_\_VA\_ARGS\_\_) split() macro(14, \_\_VA\_ARGS\_\_)

#define REPEAT\_INC\_15(macro, split, ...) REPEAT\_INC\_14(macro, split, \_\_VA\_ARGS\_\_) split() macro(15, \_\_VA\_ARGS\_\_)

#define REPEAT\_INC\_16(macro, split, ...) REPEAT\_INC\_15(macro, split, \_\_VA\_ARGS\_\_) split() macro(16, \_\_VA\_ARGS\_\_)

#define VAR\_TYPES(N) REPEAT\_INC(N, TYPENAME\_ARGS, COMMA)

#define VAR\_TYPES\_DEFAULT(N,VALUE) REPEAT\_INC(N, TYPENAME\_VALUE\_ARGS, COMMA, VALUE)

#define VAR\_ARGS(N) REPEAT\_INC(N, NAMED\_ARG, COMMA, ArgumentType)

#define VAR\_NAMED\_ARGS(N, name) REPEAT\_INC(N, NAMED\_ARG, COMMA, name)

#define VOID\_ARGS(N) REPEAT\_INC(N, VOID\_ARG, COMMA)

#define VAR\_ARGS\_MEMBERS(N, name, delim) REPEAT\_INC(N, MEMBER\_ARG, delim, name)

#define INIT\_VAR\_ARGS\_MEMBERS(N, name) REPEAT\_INC(N, INIT\_MEMBER\_ARG, COMMA, name)

#define INIT\_VALUE\_ARGS(N, name) REPEAT\_INC(N, INIT\_VALUE\_ARG, SEMICOLON, name)

#define SERIALIZE\_ARGS(N, name) REPEAT\_INC(N, SER\_ARG, SEMICOLON, name)

#define DESERIALIZE\_ARGS(N, name) REPEAT\_INC(N, DES\_ARG, SEMICOLON, name)

#define GENERATE\_METHOD\_CALLBACK\_SIGNATURES\_ARGUMENTS(N, value) \

template<typename Caller, typename ReturnType, VAR\_TYPES(N)> \

struct MethodCallbackSignature<Caller,ReturnType, VAR\_ARGS(N)> { \

typedef ReturnType(Caller::\*Method)(VAR\_ARGS(N)); }; \

template<typename Caller, VAR\_TYPES(N)> \

struct MethodCallbackSignature<Caller, void, VAR\_ARGS(N)> { \

typedef void (Caller::\*Method)(VAR\_ARGS(N)); };

#define GENERATE\_METHOD\_CALLBACK\_INVOKE\_ARGUMENTS(N, value) \

template <class BASE, class Caller, class FutureResultType, class ReturnType, VAR\_TYPES(N)> \

class MethodCallbackInvoke<BASE,Caller,FutureResultType,ReturnType, VAR\_ARGS(N)> \

: public BaseMethodCallbackInvoke<BASE, Caller, FutureResultType, ReturnType> {public: \

typedef typename MethodCallbackSignature<Caller, ReturnType, VAR\_ARGS(N)>::Method Callback; \

MethodCallbackInvoke(Caller\* caller, Callback method, VAR\_ARGS\_MEMBERS(N, arg, COMMA), const FutureResultType& result) : \

BaseMethodCallbackInvoke(caller, result), \_method(method), INIT\_VAR\_ARGS\_MEMBERS(N, arg) {} \

protected: VAR\_ARGS\_MEMBERS(N, \_arg, SEMICOLON); Callback \_method; };

#define GENERATE\_DELEGATE\_INVOKE\_ARGUMENTS(N, value) \

template <class BASE, class Caller, class FutureType, class ReturnType, VAR\_TYPES(N)> \

class DelegateInvoke<BASE, Caller, FutureType, ReturnType, VAR\_ARGS(N)> : \

public MethodCallbackInvoke<BASE, Caller, FutureType, ReturnType, VAR\_ARGS(N)> {public: \

DelegateInvoke(Caller\* caller, Callback method, VAR\_ARGS\_MEMBERS(N, arg, COMMA), const FutureType& result) \

: MethodCallbackInvoke(caller, method, VAR\_NAMED\_ARGS(N, arg), result) {} \

void invokeMethod() { \_result.resolveData(new ReturnType((\_context->\*\_method)(VAR\_NAMED\_ARGS(N, \_arg))));}}; \

template <class BASE, class Caller, class FutureType, VAR\_TYPES(N)> \

class DelegateInvoke<BASE, Caller, FutureType, void, VAR\_ARGS(N)> : \

public MethodCallbackInvoke<BASE, Caller, FutureType, void, VAR\_ARGS(N)>{public: \

DelegateInvoke(Caller\* caller, Callback method, VAR\_ARGS\_MEMBERS(N, arg, COMMA), const FutureType& result) \

: MethodCallbackInvoke(caller, method, VAR\_NAMED\_ARGS(N, arg), result) {} \

void invokeMethod() { (\_context->\*\_method)(VAR\_NAMED\_ARGS(N, \_arg));}};

#define GENERATE\_DELEGATE\_ARGUMENTS(N, value) \

template <class Caller, class ReturnType, VAR\_TYPES(N)> class Delegate<Caller, ReturnType, VAR\_ARGS(N)> : \

public DelegateInvoke < DelegateBase, Caller, FutureResult<ReturnType>, ReturnType, VAR\_ARGS(N) > { \

public: Delegate(Caller\* caller, Callback method, VAR\_ARGS\_MEMBERS(N, arg, COMMA), const FutureResult<ReturnType>& result) \

: DelegateInvoke(caller, method, VAR\_NAMED\_ARGS(N, arg), result) {} }; \

template <class Caller, VAR\_TYPES(N)> class Delegate<Caller, void, VAR\_TYPES(N)> : \

public DelegateInvoke < DelegateBase, Caller, FutureResult<void>, void, VAR\_ARGS(N) >{ \

public: Delegate(Caller\* caller, Callback method, VAR\_ARGS\_MEMBERS(N, arg, COMMA), const FutureResult<void>& result) \

: DelegateInvoke(caller, method, VAR\_NAMED\_ARGS(N, arg), result) {}};

#define GENERATE\_MULTICAST\_ARGUMENTS(N, value) \

template<class ReturnType, VAR\_TYPES(N)> \

class IDelegatePublisher<ReturnType, VAR\_ARGS(N)> {public: \

virtual FutureResult<ReturnType> publish(VAR\_ARGS\_MEMBERS(N, arg, COMMA), int priority = 0, long delay = 0, long timeout = 0) = 0; \

}; \

template <class ReturnType, VAR\_TYPES(N)> \

class MulticastDelegate<ReturnType, VAR\_ARGS(N)> : public IDelegatePublisher<ReturnType, VAR\_ARGS(N)>{private: \

typedef std::vector< IDelegatePublisher<ReturnType, VAR\_ARGS(N)>\* > SubscribersList; \

SubscribersList subscribers; \

public: \

MulticastDelegate() {} \

MulticastDelegate& operator += (IDelegatePublisher<ReturnType, VAR\_ARGS(N)>\* method) { \

subscribers.push\_back(method); \

return \*this;} \

void notify(VAR\_ARGS\_MEMBERS(N, arg, COMMA), int priority = 0, long delay = 0, long timeout = 0) { \

typename SubscribersList::iterator it = subscribers.begin(); \

for (; it != subscribers.end(); it++) { (\*it)->publish(VAR\_NAMED\_ARGS(N, arg), priority, delay, timeout); } } \

FutureResult<ReturnType> operator () (VAR\_ARGS\_MEMBERS(N, arg, COMMA), int priority = 0, long delay = 0, long timeout = 0) { \

FutureResult<ReturnType> result = publish(VAR\_NAMED\_ARGS(N, arg), priority, delay, timeout); \

notify(VAR\_NAMED\_ARGS(N, arg), priority, delay, timeout); \

return result;}};

#define GENERATE\_ACTIVE\_ARGUMENTS(N, value) \

template <class Caller, class ReturnType, VAR\_TYPES(N)> class Active<Caller, ReturnType, VAR\_ARGS(N)> : public ActiveConstraintUID, public MulticastDelegate<ReturnType, VAR\_ARGS(N)> { public: \

typedef Delegate<Caller, ReturnType, VAR\_ARGS(N)> DelegateType; \

typedef typename MethodCallbackSignature<Caller, ReturnType, VAR\_ARGS(N)>::Method Callback; \

Active(Caller\* caller, Scheduler<Caller>\* sch, Callback method) :\_context(caller), \_sch(sch), \_method(method) {} \

FutureResult<ReturnType> operator () (VAR\_ARGS\_MEMBERS(N, arg, COMMA), int priority = 0, long delay = 0, long timeout = 0) { \

FutureResult<ReturnType> result = publish(VAR\_NAMED\_ARGS(N, arg), priority, delay, timeout); return result;}; \

FutureResult<ReturnType> publish(VAR\_ARGS\_MEMBERS(N, arg, COMMA), int priority = 0, long delay = 0, long timeout = 0) { \

FutureResult<ReturnType> result(new FutureObject<ReturnType>()); \

DelegateBase::Ptr pDelegate(new DelegateType(\_context, \_method, VAR\_NAMED\_ARGS(N, arg), result)); \

\_sch->schedule(pDelegate,priority,delay, timeout, \_guardId, \_conditionId); \

this->notify(VAR\_NAMED\_ARGS(N, arg), priority, delay, timeout); \

return result;} private: Caller\* \_context; Scheduler<Caller>\* \_sch; Callback \_method; };

#define GENERATE\_METHOD\_CALLBACK\_SIGNATURES(N) REPEAT\_DEC(N, GENERATE\_METHOD\_CALLBACK\_SIGNATURES\_ARGUMENTS, EMPTY)

#define GENERATE\_METHOD\_CALLBACK\_INVOKE(N) REPEAT\_DEC(N, GENERATE\_METHOD\_CALLBACK\_INVOKE\_ARGUMENTS, EMPTY)

#define GENERATE\_DELEGATE\_INVOKE(N) REPEAT\_DEC(N, GENERATE\_DELEGATE\_INVOKE\_ARGUMENTS, EMPTY)

#define GENERATE\_DELEGATE(N) REPEAT\_DEC(N, GENERATE\_DELEGATE\_ARGUMENTS, EMPTY)

#define GENERATE\_MULTICAST\_METHOD(N) REPEAT\_DEC(N, GENERATE\_MULTICAST\_ARGUMENTS, EMPTY)

#define GENERATE\_ACTIVE\_METHOD(N) REPEAT\_DEC(N, GENERATE\_ACTIVE\_ARGUMENTS, EMPTY)

#ifdef WINDOWS\_OS

#define isnan(x) \_isnan(x)

#define isinf(x) (!\_finite(x))

#define SOCKET\_TYPE SOCKET

#define CLOSE\_SOCKET(arg) \

closesocket(arg)

#define EVENT\_TYPE HANDLE

#define CONDITION\_TYPE PLACE\_HOLDER

#define THREAD\_TYPE HANDLE

#define THREAD\_RETURN\_TYPE unsigned WINAPI

#define THREAD\_ERROR\_INSTANCE(returnValue) ((returnValue) == NULL)

#define THREAD\_ERROR\_CODE(value) GetLastError()

#define MUTEX\_CRITICAL\_SECTION CRITICAL\_SECTION

#define START\_MUTEX\_FUNCTION(arg) \

InitializeCriticalSection((arg))

#define TERMINATE\_MUTEX\_FUNCTION(arg) \

DeleteCriticalSection((arg))

#define LOCK\_MUTEX\_FUNCTION(arg) \

EnterCriticalSection((arg))

#define UNLOCK\_MUTEX\_FUNCTION(arg) \

LeaveCriticalSection((arg))

#define START\_EVENT\_TYPE\_FUNCTION(mutex, cond, reset) \

mutex = CreateEvent(NULL, reset, FALSE, NULL); \

if (!mutex) \

throw ThreadException("mutex signal failed")

#define TERMINATE\_EVENT\_FUNCTION(mutex, cond) \

CloseHandle(mutex)

#define WAIT\_EVENT\_FUNCTION(mutex, cond, wakeup) \

switch(WaitForSingleObject(mutex, INFINITE)) { \

case WAIT\_OBJECT\_0: \

return; \

default: \

throw ThreadException("wait event failed"); \

}

#define WAIT\_TIME\_EVENT\_FUNCTION(mutex, cond, time, wakeup, reset, status) \

switch (WaitForSingleObject(mutex, time + 1)) \

{ \

case WAIT\_OBJECT\_0: \

status = true; \

break; \

case WAIT\_TIMEOUT: \

status = false; \

break; \

default: \

throw ThreadException("wait failed"); \

}

#define WAKEUP\_EVENT\_FUNCTION(mutex, cond, wakeup) \

SetEvent(mutex)

#define THREAD\_JOIN\_FUNCTION(hdl) WaitForSingleObject(hdl, INFINITE)

#define THREAD\_SLEEP\_FUNCTION(ms) Sleep((ms))

#define THREAD\_CREATE\_FUNCTION(id, funPtr, callPtr) id =(HANDLE)CreateThread(NULL,0,(LPTHREAD\_START\_ROUTINE)funPtr,callPtr,0L,NULL)

#define THREAD\_TERMINATE\_FUNCTION(hdl) TerminateThread(hdl, 0)

#define THREAD\_CANCEL\_FUNCTION(hdl) TerminateThread(hdl, 0)

#define IS\_THREAD\_ALIVE\_FUNCTION(hdl, isRunning) \

DWORD exitCode = 0; \

if(GetExitCodeThread(hdl, &exitCode)) \

isRunning = (exitCode == STILL\_ACTIVE)

#define BROADCAST\_FUNCTION(arg) 0

#define SET\_EVENT\_FUNCTION(arg) \

SetEvent((arg))

#define RESET\_EVENT\_FUNCTION(arg) \

ResetEvent((arg))

#define LOCK\_MUTEX\_EVENT\_FUNCTION(arg) 0

#define UNLOCK\_MUTEX\_EVENT\_FUNCTION(arg) 0

#else

typedef int BOOL;

#ifndef FALSE

#define FALSE 0

#endif

#ifndef TRUE

#define TRUE 1

#endif

#define SOCKET\_TYPE int

#define CLOSE\_SOCKET(arg) \

close(arg)

#define EVENT\_TYPE pthread\_mutex\_t

#define CONDITION\_TYPE pthread\_cond\_t

#define THREAD\_TYPE pthread\_t

#define THREAD\_RETURN\_TYPE void \*

#define THREAD\_ERROR\_INSTANCE(returnValue) ((returnValue) == NULL)

#define THREAD\_ERROR\_CODE(value) errno

#define MUTEX\_CRITICAL\_SECTION pthread\_mutex\_t

#define START\_MUTEX\_FUNCTION(arg) \

pthread\_mutex\_init ((arg), NULL)

#define TERMINATE\_MUTEX\_FUNCTION(arg) \

pthread\_mutex\_destroy((arg))

#define LOCK\_MUTEX\_FUNCTION(arg) \

pthread\_mutex\_lock((arg))

#define UNLOCK\_MUTEX\_FUNCTION(arg) \

pthread\_mutex\_unlock((arg))

#define START\_EVENT\_TYPE\_FUNCTION(mutex, cond, reset) \

if (pthread\_mutex\_init(&mutex, NULL)) \

throw ThreadException("mutex signal failed"); \

pthread\_cond\_init(&cond, NULL)

#define TERMINATE\_EVENT\_FUNCTION(mutex, cond) \

pthread\_cond\_destroy(&cond); \

pthread\_mutex\_destroy(&mutex)

#define WAIT\_EVENT\_FUNCTION(mutex, cond, wakeup) \

pthread\_mutex\_lock(&mutex); \

int err = 0; \

while (!wakeup) { \

err = pthread\_cond\_wait(&cond, &mutex); \

if (err) { \

pthread\_mutex\_unlock(&mutex); \

throw ThreadException("wait event failed"); \

} \

} \

wakeup = FALSE; \

pthread\_mutex\_unlock(&mutex)

#define WAIT\_TIME\_EVENT\_FUNCTION(mutex, cond, ms, wakeup, reset, status) \

struct timeval tv \

struct timespec tdif \

gettimeofday(&tv, NULL) \

tdif.tv\_sec = tv.tv\_sec + ms / MUSECS\_PER\_SEC \

tdif.tv\_nsec = tv.tv\_usec\*MUSECS\_PER\_SEC + (ms % MUSECS\_PER\_SEC)\*USECS\_PER\_SEC \

if (tdif.tv\_nsec >= NSECS\_PER\_SEC) { \

tdif.tv\_nsec -= NSECS\_PER\_SEC \

tdif.tv\_sec++ \

} \

pthread\_mutex\_lock(&mutex) \

while (!wakeup) \

{ \

status = pthread\_cond\_timedwait(&cond, &mutex, &tdif) \

if(status) { \

if (status == ETIMEDOUT) break; \

pthread\_mutex\_unlock(&mutex) \

throw ThreadException(get\_error(status)) \

} \

} \

wakeup = status == 0 && reset ? false : wakeup \

pthread\_mutex\_unlock(&mutex)

#define WAKEUP\_EVENT\_FUNCTION(mutex, cond, wakeup) \

pthread\_mutex\_lock(&mutex); \

wakeup = TRUE; \

pthread\_cond\_signal(&cond); \

pthread\_mutex\_unlock(&mutex)

#define THREAD\_JOIN\_FUNCTION(id) pthread\_join(id, NULL)

#define THREAD\_SLEEP\_FUNCTION(ms) \

struct timeval tv; \

tv.tv\_usec = (ms % MUSECS\_PER\_SEC) \* USECS\_PER\_MSEC; \

tv.tv\_sec = ms / MUSECS\_PER\_SEC; \

select(0, NULL, NULL, NULL, &tv)

#define THREAD\_CREATE\_FUNCTION(id, funPtr, callPtr) \

pthread\_attr\_t attr; \

pthread\_attr\_init(&attr); \

pthread\_attr\_setdetachstate(&attr,PTHREAD\_CREATE\_DETACHED); \

pthread\_attr\_setinheritsched(&attr, PTHREAD\_INHERIT\_SCHED); \

pthread\_create(&id, &attr, funPtr, callPtr); \

pthread\_attr\_destroy(&attr)

#define THREAD\_TERMINATE\_FUNCTION(arg) pthread\_exit(arg)

#define THREAD\_CANCEL\_FUNCTION(Id) \

if (pthread\_cancel(Id) == 0) \

pthread\_detach(Id);

#define IS\_THREAD\_ALIVE\_FUNCTION(hdl, isRunning) \

isRunning = (pthread\_kill(hdl, 0) == 0)

#define BROADCAST\_FUNCTION(arg) \

pthread\_cond\_broadcast((arg))

#define SET\_EVENT\_FUNCTION(arg) 1

#define RESET\_EVENT\_FUNCTION(arg) 1

#define LOCK\_MUTEX\_EVENT\_FUNCTION(arg) \

pthread\_mutex\_lock((arg))

#define UNLOCK\_MUTEX\_EVENT\_FUNCTION(arg) \

pthread\_mutex\_unlock((arg))

#endif

#define CREATE\_THREAD(id, funPtr, callPtr) \

THREAD\_CREATE\_FUNCTION(id, funPtr, callPtr); \

if(THREAD\_ERROR\_INSTANCE(id)) \

throw ThreadException(ErrorMessage(THREAD\_ERROR\_CODE(id)))

static std::string ErrorMessage(int errorCode){

string str = "";

if (errorCode == 0) {

return str;

}

char const\* what = "Error Numer";

int whlen = strlen(what);

int ncode = errorCode, dlen = 1;

while (ncode !=0) {dlen++; ncode/=10;}

char \*buffer = (char \*) malloc(sizeof(char) \* (whlen + dlen + 1));

sprintf(buffer, "%s = %d", what, errorCode);

str = buffer;

return str;

}

class Exception : public std::exception {

friend ostream& operator<<(ostream& output, const Exception& ex) {

output << "Exception: " << ex.error;

return output;

}

public:

Exception() throw() : error(std::string("Exception")) {}

Exception(std::string err) throw() : error(err) {}

Exception(const Exception& source) throw() : std::exception(source) { error = source.error; }

virtual ~Exception() throw () {}

Exception& operator=(const Exception& source) throw() {

if (&source != this) {

error = source.error;

}

std::exception::operator= (source);

return \*this;

}

void setError(std::string exce) { error = exce; }

virtual const char\* what() const throw() { return error.c\_str(); }

protected:

std::string error;

};

struct ThreadException : public Exception{

public:

ThreadException() : Exception() {}

ThreadException(char\* errorMessage) : Exception(errorMessage) {}

ThreadException(std::string errorMessage) : Exception(errorMessage) {}

};

template <typename T>

void\* ConvertToFunctionPointer(T x) {

return \*reinterpret\_cast<void\*\*>(&x);

}

struct Runnable {

virtual void run() = 0;

};

class AtomicMutex{

public:

AtomicMutex() { START\_MUTEX\_FUNCTION(&section); }

~AtomicMutex() { TERMINATE\_MUTEX\_FUNCTION(&section); }

void lock() { LOCK\_MUTEX\_FUNCTION(&section); }

void unlock() { UNLOCK\_MUTEX\_FUNCTION(&section); }

private:

MUTEX\_CRITICAL\_SECTION section;

};

static AtomicMutex atomicMutex;

class MutexLock{

public:

MutexLock() : m\_value(0){

wakeup = FALSE;

START\_EVENT\_TYPE\_FUNCTION(mutex, cond, FALSE);

}

MutexLock(const MutexLock& m) {

this->m\_value = m.m\_value;

this->wakeup = m.wakeup;

this->mutex = m.mutex;

this->cond = m.cond;

}

~MutexLock(){

TERMINATE\_EVENT\_FUNCTION(mutex, cond);

}

void wait() {

WAIT\_EVENT\_FUNCTION(mutex, cond, wakeup);

}

void wakeUp() {

WAKEUP\_EVENT\_FUNCTION(mutex, cond, wakeup);

}

void lock() {

bool canBeAccessed = this->verifyLock();

if (canBeAccessed){

return;

}

atomicMutex.lock();

canBeAccessed = m\_value == 0;

m\_value += 1;

atomicMutex.unlock();

if (!canBeAccessed){

wait();

atomicMutex.lock();

m\_value -= 1;

atomicMutex.unlock();

}

}

bool isLocked() {

atomicMutex.lock();

bool canBeAccessed = m\_value == 0;

atomicMutex.unlock();

return !canBeAccessed;

}

bool verifyLock(int set = 0) {

atomicMutex.lock();

bool canBeAccessed = m\_value == 0;

if (m\_value == set) {

m\_value = !set;

canBeAccessed = true;

}

atomicMutex.unlock();

return canBeAccessed;

}

void unlock() {

if (!verifyLock(1)){

wakeUp();

}

}

private:

volatile int m\_value;

volatile bool wakeup;

EVENT\_TYPE mutex;

CONDITION\_TYPE cond;

};

#define synchronized(L) \

for(L.lock();L.isLocked(); \

L.unlock() )

struct ThreadParameters

{

void\* runCall;

void\* context;

ThreadParameters(void\* ctx, void\* callPtr) : context(ctx), runCall(callPtr) {}

};

class Thread: public Runnable {

public:

Thread(string threadName = "") : runnableObject(NULL), name(threadName), thrParams(this,ConvertToFunctionPointer(&Thread::run))

, \_threadHdl(0),\_isRunning(false),\_isTerminated(false) {}

Thread(Runnable \*target, string threadName = "") : name(threadName),thrParams(this,ConvertToFunctionPointer(&Thread::run)) , runnableObject(target) {}

Thread(void\* funcPtr, void\* ctx = 0 ,string threadName = "") : runnableObject(NULL), name(threadName), thrParams(ctx,funcPtr) {}

Thread(void (\*funcPtr)(void\*), void\* ctx = 0, string threadName = "") : runnableObject(NULL),name(threadName), thrParams(ctx,ConvertToFunctionPointer(funcPtr)) {}

Thread(void (\*funcPtr)(), string threadName = "") : runnableObject(NULL),name(threadName), thrParams(this,ConvertToFunctionPointer(funcPtr)) {}

template<class T>

Thread(void (T::\*RunnableCall)(), string threadName = "") : runnableObject(NULL), name(threadName), thrParams(this,ConvertToFunctionPointer(RunnableCall)) {}

template<class T>

Thread(void (T::\*RunnableCall)(void\*), void\* ctx = 0, string threadName = "") : runnableObject(NULL), name(threadName), thrParams(ctx,ConvertToFunctionPointer(RunnableCall)) {}

virtual ~Thread() {

THREAD\_TERMINATE\_FUNCTION(0);

}

static void sleep(long ms) throw(ThreadException) {

THREAD\_SLEEP\_FUNCTION(ms);

}

THREAD\_TYPE getId() const {

return this->\_threadHdl;

}

static THREAD\_RETURN\_TYPE threadFunctionPointer(void\* ptr) {

ThreadParameters\* threadParameters = (ThreadParameters\*)ptr;

if(threadParameters->context != NULL) {

Runnable\* run = (Thread\*) threadParameters->context;

run->run();

//((void (\*)(void\*)) threadParameters->runCall)(threadParameters->context);

} else {

((void (\*)(void)) threadParameters->runCall)();

}

return 0;

}

bool isFinished() {

return !this->isRunning();

}

bool isTerminated() {

synchronized(lock) {

return this->\_isTerminated;

}

return 0;

}

bool isRunning() {

synchronized(lock) {

if(this->\_isRunning) {

IS\_THREAD\_ALIVE\_FUNCTION(\_threadHdl,\_isRunning);

}

return this->\_isRunning;

}

return 0;

}

virtual void run() {

if(this->runnableObject != NULL) {

runnableObject->run();

}

}

virtual void stop() {

synchronized(lock) {

THREAD\_CANCEL\_FUNCTION(\_threadHdl);

\_isTerminated = true;

\_isRunning = false;

}

}

string getName() const {

return name;

}

void setName(string name) {

this->name = name;

}

virtual void start(Runnable \*target) throw(ThreadException) {

runnableObject = target;

start();

}

virtual void start() throw(ThreadException) {

synchronized(lock) {

reset();

CREATE\_THREAD(\_threadHdl,Thread::threadFunctionPointer, &thrParams);

\_isRunning = true;

}

}

void join() throw(ThreadException) {

THREAD\_JOIN\_FUNCTION(\_threadHdl);

}

void wait() throw(ThreadException) {

lock.wait();

}

void wakeUp() throw(ThreadException) {

lock.wakeUp();

}

private:

mutable MutexLock lock;

THREAD\_TYPE \_threadHdl;

string name;

ThreadParameters thrParams;

Runnable\* runnableObject;

bool \_isRunning;

bool \_isTerminated;

void reset() {

\_threadHdl = 0;

\_isRunning = false;

\_isTerminated = false;

}

};

template <class PT>

class AutoPtr {

public:

AutoPtr(PT\* p = NULL, bool shared = false) : \_ptr(NULL) { \_ptr=p; if (shared && \_ptr) \_ptr->reference(); }

AutoPtr(const AutoPtr& ptr) : \_ptr(NULL) { \_ptr=ptr.\_ptr; if (\_ptr) \_ptr->reference(); }

~AutoPtr() { if (\_ptr) \_ptr->release(); }

AutoPtr& operator = (const AutoPtr& ptr) {

if (&ptr != this) {

if (\_ptr) \_ptr->release();

\_ptr = ptr.\_ptr;

if (\_ptr) \_ptr->reference();

}

return \*this;

}

PT\* operator -> () {

if (\_ptr)

return \_ptr;

else

throw std::exception("Null Pointer Exception");

}

PT\* reference() { if (\_ptr) \_ptr->reference(); return \_ptr; }

PT\* value() { return \_ptr; }

operator const PT\* () const { return \_ptr; }

bool operator == (const AutoPtr& ptr) const { return \_ptr == ptr.\_ptr; }

bool operator != (const AutoPtr& ptr) const { return \_ptr != ptr.\_ptr; }

private:

PT\* \_ptr;

};

class ReferenceObject {

private:

mutable MutexLock mutex;

mutable volatile int referenceNumber;

public:

ReferenceObject() {

synchronized(mutex) {

referenceNumber = 1;

}

}

void reference() const {

synchronized(mutex) {

++referenceNumber;

}

}

void release() const {

synchronized(mutex) {

--referenceNumber;

}

if (referenceNumber == 0) {

delete this;

}

}

size\_t size() const {

int val = 0;

synchronized(mutex) {

val = referenceNumber;

}

}

protected:

virtual ~ReferenceObject() {}

};

template <typename Object> struct RefPointer : public ReferenceObject { typedef AutoPtr<Object> Ptr; };

class AutoLock {

public:

explicit AutoLock(MutexLock& mutex) : \_mutex(mutex) { \_mutex.lock(); }

~AutoLock() { try{ \_mutex.unlock(); } catch (...) {} }

private:

MutexLock& \_mutex;

};

class Signal {

public:

Signal(bool manualReset = true) {

START\_EVENT\_TYPE\_FUNCTION(\_mutex, \_cond, manualReset ? FALSE : TRUE);

}

~Signal() {

TERMINATE\_EVENT\_FUNCTION(\_mutex, \_cond);

}

void notify() {

if (LOCK\_MUTEX\_EVENT\_FUNCTION(&\_mutex))

throw ThreadException("cannot notify lock");

if (BROADCAST\_FUNCTION(&\_cond))

{

UNLOCK\_MUTEX\_EVENT\_FUNCTION(&\_mutex);

throw ThreadException("cannot notify lock");

}

if (!SET\_EVENT\_FUNCTION(\_mutex))

{

throw ThreadException("cannot notify lock");

}

UNLOCK\_MUTEX\_EVENT\_FUNCTION(&\_mutex);

}

void wait() {

WAIT\_EVENT\_FUNCTION(\_mutex, \_cond, \_state);

}

bool wait(long ms, bool timeout = false) {

int status = false;

WAIT\_TIME\_EVENT\_FUNCTION(\_mutex, \_cond, ms, \_state, \_auto, status)

if (timeout && !status)

throw ThreadException("Timeout Exception");

return status;

}

void reset()

{

if (LOCK\_MUTEX\_EVENT\_FUNCTION(&\_mutex)) {

throw ThreadException("reset signal lock");

}

if (!RESET\_EVENT\_FUNCTION(\_mutex)) {

throw ThreadException("reset signal lock");

}

UNLOCK\_MUTEX\_EVENT\_FUNCTION(&\_mutex);

}

private:

EVENT\_TYPE \_mutex;

CONDITION\_TYPE \_cond;

};

class UID {

public:

unsigned int \_uid;

UID() { \_uid = ++uid; }

UID(const UID& uid) { \_uid = uid.\_uid; }

UID& operator=(const UID& uid) { \_uid = uid.\_uid; return(\*this); }

operator int() { return \_uid; }

bool operator == (const UID& uid) const { return \_uid == uid.\_uid; }

bool operator != (const UID& uid) const { return \_uid != uid.\_uid; }

bool operator == (const unsigned int& uid) const { return \_uid == uid; }

bool operator != (const unsigned int& uid) const { return \_uid != uid; }

protected:

static unsigned int uid;

};

template <class DataType>

class DataResolver {

public:

DataResolver() : \_data(0) { }

DataType& data() {

return \*\_data;

}

void resolveData(DataType\* data) {

delete \_data;

\_data = data;

}

private:

DataType\* \_data;

};

class ErrorResolver {

public:

ErrorResolver() :\_error(0) {}

std::string getErrorMessage() const {

return (\_error) ? \_error->what() : std::string();

}

std::exception\* getError() const {

return \_error;

}

void resolveError(const std::string& msg) {

delete \_error;

\_error = new std::exception(msg.c\_str());

}

bool hasError() const {

return \_error != 0;

}

private:

std::exception\* \_error;

};

class SharedObject : public ReferenceObject {

public:

SharedObject() : \_signal(false) {}

void wait() {

\_signal.wait();

}

bool wait(long ms, bool timeout = false) {

return \_signal.wait(ms, timeout);

}

void notify() {

\_signal.notify();

}

private:

Signal \_signal;

};

template <typename T>

struct SharedObjectProxy

{

typedef T\* DataTypePtr;

public:

SharedObjectProxy(DataTypePtr data) : \_data(data) {}

SharedObjectProxy(const SharedObjectProxy& proxy) {

\_data = proxy.\_data;

\_data->reference();

}

~SharedObjectProxy() {

\_data->release();

}

void snooze() {

\_data->wait(0); //for (; data->wait(10); );

}

void wait() {

\_data->wait();

}

bool wait(long ms, bool timeout = false) {

return \_data->wait(ms, timeout);

}

bool ready() const {

return \_data->wait(0);

}

void notify() {

\_data->notify();

}

protected:

SharedObjectProxy();

DataTypePtr \_data;

};

template <typename BASE, typename T = typename BASE::DataTypePtr>

struct ErrorProxy : public BASE {

public:

ErrorProxy(T data) : BASE(data) {}

ErrorProxy(const ErrorProxy& proxy) : BASE(proxy) {}

std::string getErrorMessage() const {

return \_data->getErrorMessage();

}

std::exception\* getError() const {

return \_data->getError();

}

void resolveError(const std::string& msg) {

\_data->resolveError(msg);

}

bool hasError() const {

return \_data->hasError();

}

};

template <typename BASE, class Type, typename T = typename BASE::DataTypePtr>

struct DataProxy : public BASE {

public:

DataProxy(T data) : BASE(data) {}

DataProxy(const DataProxy& proxy) : BASE(proxy) {}

Type& data() const {

return \_data->data();

}

void resolveData(Type\* data) {

\_data->resolveData(data);

}

};

template<typename Caller, typename ReturnType = void, VAR\_TYPES\_DEFAULT(ARGUMENT\_UPPER\_LIMIT, void)> struct MethodCallbackSignature;

template<typename Caller, typename ReturnType, VAR\_TYPES(ARGUMENT\_UPPER\_LIMIT)> struct MethodCallbackSignature { typedef ReturnType(Caller::\*Method)(VAR\_ARGS(ARGUMENT\_UPPER\_LIMIT)); };

template<typename Caller, typename ReturnType> struct MethodCallbackSignature<Caller, ReturnType, VOID\_ARGS(ARGUMENT\_UPPER\_LIMIT)> { typedef ReturnType(Caller::\*Method)(); };

template<typename Caller, VAR\_TYPES(ARGUMENT\_UPPER\_LIMIT)> struct MethodCallbackSignature<Caller, void, VAR\_ARGS(ARGUMENT\_UPPER\_LIMIT)> { typedef void (Caller::\*Method)(VAR\_ARGS(ARGUMENT\_UPPER\_LIMIT)); };

template<typename Caller> struct MethodCallbackSignature<Caller, void, VOID\_ARGS(ARGUMENT\_UPPER\_LIMIT)> { typedef void (Caller::\*Method)(); };

GENERATE\_METHOD\_CALLBACK\_SIGNATURES(ARGUMENT\_UPPER\_LIMIT)

template <class BASE, class Caller, class FutureType, class ReturnType>

class BaseMethodCallbackInvoke : public BASE {

public:

BaseMethodCallbackInvoke(Caller\* caller, const FutureType& result) :

\_result(result), \_context(caller) {}

~BaseMethodCallbackInvoke() { this->release(); }

void run(){

try {

invokeMethod();

}

catch (std::exception& e) {

\_result.resolveError(e.what());

}

catch (...) {

\_result.resolveError("Invoke Error");

}

\_result.notify();

}

void resolveError(const std::string& msg) {

\_result.resolveError(msg);

}

protected:

virtual void invokeMethod() = 0;

Caller\* \_context;

FutureType \_result;

};

template<class BASE, class Caller, class FutureType, class ReturnType, VAR\_TYPES\_DEFAULT(ARGUMENT\_UPPER\_LIMIT, void)> class MethodCallbackInvoke;

template <class BASE, class Caller, class FutureType, class ReturnType = void, VAR\_TYPES(ARGUMENT\_UPPER\_LIMIT)>

class MethodCallbackInvoke : public BaseMethodCallbackInvoke<BASE, Caller, FutureType, ReturnType> {

public:

typedef typename MethodCallbackSignature<Caller, ReturnType, VAR\_ARGS(ARGUMENT\_UPPER\_LIMIT)>::Method Callback;

MethodCallbackInvoke(Caller\* caller, Callback method, VAR\_ARGS\_MEMBERS(ARGUMENT\_UPPER\_LIMIT, arg, COMMA), const FutureType& result) :

BaseMethodCallbackInvoke(caller, result), \_method(method), INIT\_VAR\_ARGS\_MEMBERS(ARGUMENT\_UPPER\_LIMIT, arg) {}

protected:

VAR\_ARGS\_MEMBERS(ARGUMENT\_UPPER\_LIMIT, \_arg, SEMICOLON);

Callback \_method;

};

GENERATE\_METHOD\_CALLBACK\_INVOKE(ARGUMENT\_UPPER\_LIMIT)

template <class BASE, class Caller, class FutureType, class ReturnType>

class MethodCallbackInvoke<BASE, Caller, FutureType, ReturnType> : public BaseMethodCallbackInvoke<BASE, Caller, FutureType, ReturnType>{

public:

typedef typename MethodCallbackSignature<Caller, ReturnType>::Method Callback;

MethodCallbackInvoke(Caller\* caller, Callback method, const FutureType& result) :

BaseMethodCallbackInvoke(caller, result), \_method(method) {}

protected:

Callback \_method;

};

template<class BASE, class Caller, class FutureType, class ReturnType = void, VAR\_TYPES\_DEFAULT(ARGUMENT\_UPPER\_LIMIT, void)> class DelegateInvoke;

template <class BASE, class Caller, class FutureType, class ReturnType, VAR\_TYPES(ARGUMENT\_UPPER\_LIMIT)>

class DelegateInvoke : public MethodCallbackInvoke<BASE, Caller, FutureType, ReturnType, VAR\_ARGS(ARGUMENT\_UPPER\_LIMIT)> {

public:

DelegateInvoke(Caller\* caller, Callback method, VAR\_ARGS\_MEMBERS(ARGUMENT\_UPPER\_LIMIT, arg, COMMA), const FutureType& result)

: MethodCallbackInvoke(caller, method, VAR\_NAMED\_ARGS(N, arg), result) {}

void invokeMethod() {

\_result.resolveData(new ReturnType((\_context->\*\_method)(VAR\_ARGS(ARGUMENT\_UPPER\_LIMIT))));

}

};

template <class BASE, class Caller, class FutureType, VAR\_TYPES(ARGUMENT\_UPPER\_LIMIT)>

class DelegateInvoke<BASE, Caller, FutureType, void, VAR\_ARGS(ARGUMENT\_UPPER\_LIMIT)> : public MethodCallbackInvoke<BASE, Caller, FutureType, void, VAR\_ARGS(ARGUMENT\_UPPER\_LIMIT)>{

public:

DelegateInvoke(Caller\* caller, Callback method, VAR\_ARGS\_MEMBERS(ARGUMENT\_UPPER\_LIMIT, arg, COMMA), const FutureType& result)

: MethodCallbackInvoke(caller, method, VAR\_NAMED\_ARGS(N, arg), result) {}

void invokeMethod() {

(\_context->\*\_method)(VAR\_ARGS(ARGUMENT\_UPPER\_LIMIT));

}

};

GENERATE\_DELEGATE\_INVOKE(ARGUMENT\_UPPER\_LIMIT)

template <class BASE, class Caller, class FutureType, class ReturnType>

class DelegateInvoke<BASE, Caller, FutureType, ReturnType> : public MethodCallbackInvoke<BASE, Caller, FutureType, ReturnType, void>{

public:

DelegateInvoke(Caller\* caller, Callback method, const FutureType& result) : MethodCallbackInvoke(caller, method, result) {}

void invokeMethod() {

\_result.resolveData(new ReturnType((\_context->\*\_method)()));

}

};

template <class BASE, class Caller, class FutureType>

class DelegateInvoke<BASE, Caller, FutureType> : public MethodCallbackInvoke<BASE, Caller, FutureType, void, void>{

public:

DelegateInvoke(Caller\* caller, Callback method, const FutureType& result) : MethodCallbackInvoke(caller, method, result) {}

void invokeMethod() {

(\_context->\*\_method)();

}

};

class DelegateBase : public Runnable, public RefPointer<DelegateBase> {

public:

virtual void resolveError(const std::string& msg) = 0;

};

template <class FutureType> class FutureObject : public SharedObject, public DataResolver<FutureType>, public ErrorResolver {};

template <> class FutureObject<void> : public SharedObject, public ErrorResolver{};

template <class FutureType> class FutureResult : public DataProxy< ErrorProxy< SharedObjectProxy<FutureObject<FutureType>>>, FutureType>{

public: FutureResult(SharedObjectProxy::DataTypePtr ptr) :DataProxy(ptr){}

};

template <> class FutureResult<void> : public ErrorProxy< SharedObjectProxy<FutureObject<void>>>{

public: FutureResult(SharedObjectProxy::DataTypePtr ptr) :ErrorProxy(ptr){}

};

template<class Caller, class ReturnType = void, VAR\_TYPES\_DEFAULT(ARGUMENT\_UPPER\_LIMIT, void)> class Delegate;

template <class Caller, class ReturnType, VAR\_TYPES(ARGUMENT\_UPPER\_LIMIT)> class Delegate : public DelegateInvoke < DelegateBase, Caller, FutureResult<ReturnType>, ReturnType, VAR\_ARGS(ARGUMENT\_UPPER\_LIMIT) > {

public: Delegate(Caller\* caller, Callback method, VAR\_ARGS\_MEMBERS(ARGUMENT\_UPPER\_LIMIT, arg, COMMA), const FutureResult<ReturnType>& result) : DelegateInvoke(caller, method, VAR\_NAMED\_ARGS(N, arg), result) {}

};

template <class Caller, VAR\_TYPES(ARGUMENT\_UPPER\_LIMIT)> class Delegate<Caller, void, VAR\_TYPES(ARGUMENT\_UPPER\_LIMIT)> : public DelegateInvoke < DelegateBase, Caller, FutureResult<void>, void, VAR\_ARGS(ARGUMENT\_UPPER\_LIMIT) >{

public: Delegate(Caller\* caller, Callback method, VAR\_ARGS\_MEMBERS(ARGUMENT\_UPPER\_LIMIT, arg, COMMA), const FutureResult<void>& result) : DelegateInvoke(caller, method, VAR\_NAMED\_ARGS(N, arg), result) {}

};

template <class Caller, class ReturnType> class Delegate<Caller, ReturnType, void> : public DelegateInvoke< DelegateBase, Caller, FutureResult<ReturnType>, ReturnType, void >{

public: Delegate(Caller\* caller, Callback method, const FutureResult<ReturnType>& result) : DelegateInvoke(caller, method, result) {}

};

template <class Caller> class Delegate<Caller, void, void> : public DelegateInvoke< DelegateBase, Caller, FutureResult<void>, void, void >{

public: Delegate(Caller\* caller, Callback method, const FutureResult<void>& result) : DelegateInvoke(caller, method, result) {}

};

GENERATE\_DELEGATE(ARGUMENT\_UPPER\_LIMIT)

template <class Caller>

struct Request : public RefPointer<Request<Caller>> {

public:

typedef std::deque<Ptr> RequestsQueue;

typedef typename MethodCallbackSignature<Caller, bool, int>::Method RequestGuard;

Request(int priority = 0, long delay = 0, long timeout = 0, Caller\* caller = 0, RequestGuard guard = 0, int guardId = 0, int conditionId = 0) : \_priority(priority), \_delay(delay), \_timeout(timeout), \_context(caller), \_guard(guard), \_guardId(guardId), \_conditionId(conditionId) {}

int getPriority() const { return \_priority; }

int getDelay() const { return \_delay; }

int getTimeout() const { return \_timeout; }

bool filtered() {

if (\_guard){

return !((\_context->\*\_guard)(\_guardId));

}

return false;

}

bool deferred() {

if (\_guard){

return !((\_context->\*\_guard)(\_conditionId));

}

return false;

}

private:

Caller\* \_context;

RequestGuard \_guard;

int \_guardId;

int \_conditionId;

int \_priority;

long \_delay;

long \_timeout;

};

template <class Caller>

struct MethodAccessRequest {

typename Request<Caller>::Ptr request;

Signal ready;

typedef std::deque<MethodAccessRequest<Caller>\*> MethodAccessRequestQueue;

};

template <class Caller> struct MethodInvokeRequest : public Request<Caller> {

public:

MethodInvokeRequest(DelegateBase::Ptr requestRunnable, int priority = 0, long delay = 0, long timeout = 0, Caller\* caller = 0, RequestGuard guard = 0, int guardId = 0, int conditionId = 0)

: \_requestRunnable(requestRunnable), Request(priority, delay, timeout, caller, guard, guardId, conditionId){}

DelegateBase::Ptr getRequest() const { return \_requestRunnable; }

private:

DelegateBase::Ptr \_requestRunnable;

};

template <class Caller>

class ScheduleQueue : protected Runnable {

public:

ScheduleQueue() : deferringRunning(false) {}

~ScheduleQueue() { try { clear(); } catch (...) {} }

void request(typename Request<Caller>::Ptr request, bool urgent = false) {

synchronized(\_lock) {

deferringRunning = false;

if (request->deferred()) {

\_deferred.push\_back(request);

}

else {

addRequest(request, urgent);

}

}

}

typename Request<Caller>\* processPendingRequests(){

//::Sleep(1); // DEBUG

Request<Caller>::Ptr request = NULL;

MethodAccessRequest<Caller>\* mq = NULL;

processDeferred();

synchronized(\_lock){

// ::Sleep(1); // DEBUG

request = getNextRequest();

}

if (request.value() != NULL) {

return request.reference();

}

else if (\_pending.empty() && !\_deferred.empty() && !deferringRunning) {

\_thread.start(this);

deferringRunning = true;

}

mq = new MethodAccessRequest<Caller>();

\_pending.push\_back(mq);

mq->ready.wait();

request = mq->request;

delete mq;

return request.reference();

}

void done() {

synchronized(\_lock) {

for (MethodAccessRequest<Caller>::MethodAccessRequestQueue::iterator it = \_pending.begin(); it != \_pending.end(); ++it) {

(\*it)->ready.notify();

}

\_pending.clear();

}

}

bool empty() const {

synchronized(\_lock) {

return \_requests.empty() && \_deferred.empty();

}

}

int size() const {

synchronized(\_lock) {

return static\_cast<int>(\_requests.size() + \_deferred.size());

}

}

void clear() {

synchronized(\_lock) {

\_requests.clear();

\_deferred.clear();

}

\_thread.join();

}

protected:

void run() {

while (deferringRunning) { try { processDeferred(); } catch (...) {} }

}

private:

void addRequest(typename Request<Caller>::Ptr request, bool urgent = false) {

if (\_pending.empty()) {

if (urgent) {

\_requests.push\_front(request);

}

else if (request.value() && request->getPriority() <= 0) {

\_requests.push\_back(request);

}

else {

typename Request<Caller>::RequestsQueue::iterator it;

for (it = \_requests.begin(); it != \_requests.end(); ++it) {

if (request->getPriority() > (\*it)->getPriority())

break;

}

\_requests.insert(it, request);

}

}

else {

MethodAccessRequest<Caller>\* mq = \_pending.front();

\_pending.pop\_front();

mq->request = request;

mq->ready.notify();

}

}

void processDeferred() {

if (!\_deferred.empty()) {

Request<Caller>::Ptr deferred = \_deferred.front();

if (deferred.value() && !deferred->deferred()) {

\_deferred.pop\_front();

addRequest(deferred);

}

}

else {

deferringRunning = false;

return;

}

}

typename Request<Caller>::Ptr getNextRequest() {

Request<Caller>::Ptr request;

if (!\_requests.empty()) {

request = \_requests.front();

\_requests.pop\_front();

}

return request;

}

private:

mutable MutexLock \_lock;

Thread \_thread;

bool deferringRunning;

typename Request<Caller>::RequestsQueue \_requests;

typename Request<Caller>::RequestsQueue \_deferred;

typename MethodAccessRequest<Caller>::MethodAccessRequestQueue \_pending;

};

class TimedEvent : protected Runnable {

public:

TimedEvent() { }

virtual ~TimedEvent() { }

void timeout(DelegateBase::Ptr method, long ms) {

\_method = method;

\_thread.start(this);

try{

\_timeoutEvent.wait(ms, true);

}

catch (...) {

\_thread.stop();

\_method->resolveError("Timeout Exception");

\_method = NULL;

\_timeoutEvent.notify();

}

}

protected:

void run() {

\_method->reference();

\_method->run();

\_timeoutEvent.notify();

}

private:

Thread \_thread;

DelegateBase::Ptr \_method;

Signal \_timeoutEvent;

};

template <class Caller>

class Scheduler : protected Runnable {

public:

typedef typename Request<Caller>::RequestGuard GuardList;

Scheduler(Caller\* caller = 0, GuardList guardList = 0) : \_context(caller), \_guardList(guardList) {

\_run = true;

\_thread.start(this);

}

virtual ~Scheduler() { try { stop(); } catch (...) {} }

void schedule(DelegateBase::Ptr pDelegate, int priority, long delay, long timeout, UID guard, UID condition) {

\_queue.request(new MethodInvokeRequest<Caller>(pDelegate, priority, delay, timeout, \_context, \_guardList, guard, condition));

}

void cancel() {

\_queue.clear();

}

protected:

void run() {

AutoPtr<Request<Caller>> pendingRequest = \_queue.processPendingRequests();

while (pendingRequest) {

MethodInvokeRequest<Caller>\* mth = dynamic\_cast<MethodInvokeRequest<Caller>\*>(pendingRequest.value());

if (mth != NULL && !mth->filtered()) {

long delay = mth->getDelay();

if (delay > 0) {

Thread::sleep(delay);

}

if (mth->getTimeout() > 0) {

\_timeEvent.timeout(mth->getRequest(), mth->getTimeout());

}

else {

DelegateBase::Ptr pDelegate = mth->getRequest();

if (pDelegate.value()) {

pDelegate->reference();

pDelegate->run();

pDelegate = NULL;

}

}

}

pendingRequest = NULL;

if (\_run)

pendingRequest = \_queue.processPendingRequests();

}

}

void stop() {

\_queue.clear();

\_queue.done();

\_run = false;

\_thread.join();

}

private:

Caller\* \_context;

Thread \_thread;

ScheduleQueue<Caller> \_queue;

GuardList \_guardList;

bool \_run;

TimedEvent \_timeEvent;

};

struct ActiveConstraintUID {

public:

UID getGuardUID() { return \_guardId; }

UID getConditionUID() { return \_conditionId; }

protected:

UID \_guardId;

UID \_conditionId;

};

template<class ReturnType, VAR\_TYPES\_DEFAULT(ARGUMENT\_UPPER\_LIMIT, void)> class IDelegatePublisher;

template<class ReturnType, VAR\_TYPES(ARGUMENT\_UPPER\_LIMIT)>

class IDelegatePublisher {

public:

virtual FutureResult<ReturnType> publish(VAR\_ARGS\_MEMBERS(ARGUMENT\_UPPER\_LIMIT, arg, COMMA), int priority = 0, long delay = 0, long timeout = 0) = 0;

};

template<class ReturnType>

class IDelegatePublisher<ReturnType> {

public:

virtual FutureResult<ReturnType> publish(int priority = 0, long delay = 0, long timeout = 0) = 0;

};

template<class ReturnType, VAR\_TYPES\_DEFAULT(ARGUMENT\_UPPER\_LIMIT, void)> class MulticastDelegate;

template <class ReturnType, VAR\_TYPES(ARGUMENT\_UPPER\_LIMIT)>

class MulticastDelegate : public IDelegatePublisher<ReturnType, VAR\_ARGS(ARGUMENT\_UPPER\_LIMIT)> {

private:

typedef std::vector< IDelegatePublisher<ReturnType, VAR\_TYPES(ARGUMENT\_UPPER\_LIMIT)> \* > SubscribersList;

SubscribersList subscribers;

public:

MulticastDelegate() {}

MulticastDelegate& operator += (const IDelegatePublisher<ReturnType, VAR\_TYPES(ARGUMENT\_UPPER\_LIMIT)>\* method) {

subscribers.push\_back(method);

return \*this;

}

void notify(VAR\_ARGS\_MEMBERS(ARGUMENT\_UPPER\_LIMIT, arg, COMMA), int priority = 0, long delay = 0, long timeout = 0) {

typename SubscribersList::iterator it = subscribers.begin();

for (; it != subscribers.end(); it++) { it->publish(VAR\_NAMED\_ARGS(ARGUMENT\_UPPER\_LIMIT, arg), priority, delay, timeout); }

}

FutureResult<ReturnType> operator () (VAR\_ARGS\_MEMBERS(ARGUMENT\_UPPER\_LIMIT, arg, COMMA), int priority = 0, long delay = 0, long timeout = 0) {

FutureResult<ReturnType> result = publish(VAR\_NAMED\_ARGS(ARGUMENT\_UPPER\_LIMIT, arg), priority, delay, timeout);

notify(VAR\_NAMED\_ARGS(ARGUMENT\_UPPER\_LIMIT, arg), priority, delay, timeout);

return result;

}

};

template <class ReturnType>

class MulticastDelegate<ReturnType> : public IDelegatePublisher<ReturnType>{

private:

typedef std::vector< IDelegatePublisher<ReturnType> \* > SubscribersList;

SubscribersList subscribers;

public:

MulticastDelegate() {}

MulticastDelegate& operator += (const IDelegatePublisher<ReturnType>\* method) {

subscribers.push\_back(method);

return \*this;

}

void notify(int priority = 0, long delay = 0, long timeout = 0) {

typename SubscribersList::iterator it = subscribers.begin();

for (; it != subscribers.end(); it++) { (\*it)->publish(priority, delay, timeout); }

}

FutureResult<ReturnType> operator () (int priority = 0, long delay = 0, long timeout = 0) {

FutureResult<ReturnType> result = publish(priority, delay, timeout);

notify(priority, delay, timeout);

return result;

}

};

GENERATE\_MULTICAST\_METHOD(ARGUMENT\_UPPER\_LIMIT)

template<class Caller, class ReturnType, VAR\_TYPES\_DEFAULT(ARGUMENT\_UPPER\_LIMIT, void)> class Active;

template <class Caller, class ReturnType, VAR\_TYPES(ARGUMENT\_UPPER\_LIMIT)>

class Active : public ActiveConstraintUID, public MulticastDelegate<ReturnType, VAR\_ARGS(ARGUMENT\_UPPER\_LIMIT)>{

public:

typedef Delegate<Caller, ReturnType, VAR\_ARGS(ARGUMENT\_UPPER\_LIMIT)> DelegateType;

typedef typename MethodCallbackSignature<Caller, ReturnType, VAR\_ARGS(ARGUMENT\_UPPER\_LIMIT)>::Method Callback;

Active(Caller\* caller, Scheduler<Caller>\* sch, Callback method) :\_context(caller), \_sch(sch), \_method(method) {}

FutureResult<ReturnType> operator () (VAR\_ARGS\_MEMBERS(ARGUMENT\_UPPER\_LIMIT, arg, COMMA), int priority = 0, long delay = 0, long timeout = 0) {

FutureResult<ReturnType> result = publish(priority, delay, timeout);

return result;

}

FutureResult<ReturnType> publish(VAR\_ARGS\_MEMBERS(ARGUMENT\_UPPER\_LIMIT, arg, COMMA), int priority = 0, long delay = 0, long timeout = 0) {

FutureResult<ReturnType> result(new FutureObject<ReturnType>());

DelegateBase::Ptr pDelegate(new DelegateType(\_context, \_method, VAR\_NAMED\_ARGS(ARGUMENT\_UPPER\_LIMIT, arg), result));

\_sch->schedule(pDelegate, priority, delay, timeout, \_guardId, \_conditionId);

this->notify(VAR\_NAMED\_ARGS(ARGUMENT\_UPPER\_LIMIT, arg), priority, delay, timeout);

return result;

}

private:

Caller\* \_context;

Scheduler<Caller>\* \_sch;

Callback \_method;

};

template <class Caller, class ReturnType>

class Active <Caller, ReturnType, void> : public ActiveConstraintUID, public MulticastDelegate<ReturnType> {

public:

typedef Delegate<Caller, ReturnType, void> DelegateType;

typedef typename MethodCallbackSignature<Caller, ReturnType>::Method Callback;

Active(Caller\* caller, Scheduler<Caller>\* sch, Callback method) :\_context(caller), \_sch(sch), \_method(method) {}

FutureResult<ReturnType> publish(int priority = 0, long delay = 0, long timeout = 0) {

FutureResult<ReturnType> result(new FutureObject<ReturnType>());

DelegateBase::Ptr pDelegate(new DelegateType(\_context, \_method, result));

\_sch->schedule(pDelegate, priority, delay, timeout, \_guardId, \_conditionId);

this->notify(priority, delay, timeout);

return result;

}

private:

Caller\* \_context;

Scheduler<Caller>\* \_sch;

Callback \_method;

};

GENERATE\_ACTIVE\_METHOD(ARGUMENT\_UPPER\_LIMIT)

template <class Caller>

class AsyncMethod : protected Runnable, public ActiveConstraintUID {

public:

typedef Delegate<Caller, void, void> DelegateType;

typedef Active<Caller, void> ActiveType;

typedef typename MethodCallbackSignature<Caller>::Method Callback;

AsyncMethod(Caller\* caller, Callback method, long interval = 0, long delay = 0, long timeout = 0) :\_context(caller), \_delegate(new DelegateType(\_context, method, new FutureObject<void>())), \_delay(delay), \_timeout(timeout), \_interval(interval), \_active(false){}

AsyncMethod(Caller\* caller, Scheduler<Caller>\* sch, Callback method, long interval = 0, long delay = 0, long timeout = 0) :\_context(caller), \_delegate(new DelegateType(\_context, method, new FutureObject<void>())), \_delay(delay), \_timeout(timeout), \_interval(interval), \_sch(sch), \_active(true) {}

virtual ~AsyncMethod() { try { stop(); \_thread.join(); } catch (...) {} }

void start() {

\_run = true;

\_thread.start(this);

}

void stop() {

\_run = false;

}

protected:

void run() {

while (\_run) {

if (!\_active && \_delay > 0) {

Thread::sleep(\_delay);

}

if (\_interval > 0) {

if (\_active) {

\_sch->schedule(\_delegate, -1, \_delay, \_timeout, \_guardId, \_conditionId);

}

else {

\_delegate->run();

if (\_timeout > 0)

\_timeoutEvent.wait(\_timeout, true);

}

Thread::sleep(\_interval);

}

}

}

private:

bool \_run;

long \_interval;

long \_delay;

long \_timeout;

bool \_active;

Thread \_thread;

Caller\* \_context;

DelegateBase::Ptr \_delegate;

Scheduler<Caller>\* \_sch;

Signal \_timeoutEvent;

};

//A template function to get the index of an item for a given vector. It returns -1 if the

//item was not found

template <typename InputIterator, typename EqualityComparable>

int IndexOf(const InputIterator& begin,

const InputIterator& end, const EqualityComparable& item) {

if(begin == end )

return -1;

InputIterator fnd = std::find(begin, end, item);

unsigned int index = std::distance(begin, fnd);

return fnd != end ? index : -1;

};

//Used as a function pointer to safely destroy elements or collections of elements

//Example:

// std::for\_each( arr->begin(), arr->end(), delete\_pointer\_element());

// delete arr;

struct deleteElement{

template< typename T >

void operator()( T element ) const{

delete element;

}

};

template<typename T>

unsigned int compare(T c1, T c2, unsigned int sz ) {

if(!is\_ptr<T>::value) {

return std::memcmp(&c1,&c2, sz);

}

return std::memcmp((void\*)c1, (void\*)c2, sz);

};

template<typename T>

void copyObject(T\* object1, T\* object2, unsigned int sz ) {

if(!is\_ptr<T>::value) {

std::memcpy(&object2,&object1, sz);

}

std::memcpy((void\*)object2, (void\*)object1, sz);

};

//A template function to copy vector

template<typename InputIterator, typename T>

void copyVector(InputIterator& it, InputIterator& end ,T\* vectorObj) {

for (; it != end; ++it) {

vectorObj->push\_back(\*it);

}

};

#endif

//%% NEW FILE User.cpp BEGINS HERE %%

//PLEASE DO NOT EDIT THIS CODE

//This code was generated using the UMPLE 1.29.1.4448.81a70243a modeling language

#define DEF\_\_USER\_BODY

#if defined( PRAGMA ) && ! defined( PRAGMA\_IMPLEMENTED )

#pragma implementation <User.h>

#endif

//------------------------

//FILE INCLUDES

//------------------------

#include <model\_Model.h>

#include <User.h>

#include <Stand.h>

#include <Vote.h>

#include <Statistics.h>

//------------------------

//LIBRARY INCLUDES

//------------------------

using namespace std;

#include <ostream>

#include <vector>

//------------------------

//CONSTRUCTOR IMPLEMENTATION

//------------------------

User::User(const string aId, Stand\* aStand):

stand(NULL){

this->id= aId;

this->votes= new vector<Vote\*>();

if (!setStand(aStand)){

throw "Unable to create User due to aStand";

}

}

User::User(User& other):

stand(NULL){

internalCopy(other);

}

//------------------------

//STREAM HELPER GROUP IMPLEMENTATION

//------------------------

void User::toOstream(ostream& os) const{

User\* thisptr = const\_cast<User\*>(this);

os << "[" << "id:" << thisptr->getId() << "]" << endl;

(thisptr->getVotes()!= NULL ? os << "votes:" << thisptr->getVotes() : os << "votes:" << "NULL")<< endl;

(thisptr->getStand()!= NULL ? os << "stand:" << thisptr->getStand() : os << "stand:" << "NULL");

}

ostream& operator<<(ostream& os, const User& dt){

dt.toOstream(os);

return os;

}

//------------------------

//PREDEFINED OPERATORS IMPLEMENTATION

//------------------------

bool operator == (User& Right, User& Left){

//if (typeid(Right) != typeid(Left)) {

// return false;

//}

if(Right.hashCode() != Left.hashCode()) {

return false;

}

if(Right.id.compare(Left.id) != 0){

return false;

}

if(!std::equal(Right.votes->begin(), Right.votes->end(), Left.votes->begin())){

return false;

}

if(!compare(Right.stand, Left.stand, sizeof Right.stand)){

return false;

}

}

User& User::operator=(User& other){

internalCopy(other);

return \*this;

}

void User::internalCopy(User& other){

this->id.assign(other.id);

this->votes = new vector<Vote\*>();

vector<Vote\*>::iterator votesBeginIterator(other.votes->begin());

vector<Vote\*>::iterator votesEndIterator(other.votes->end());

copyVector(votesBeginIterator, votesEndIterator, this->votes);

copyObject(other.stand, this->stand, sizeof other.stand);

}

bool User::setId(const string aNewId){

bool wasSet= false;

this->id = aNewId;

wasSet= true;

return wasSet;

}

bool User::setStand(Stand\* aNewStand){

bool wasSet= false;

if (aNewStand == NULL){

return wasSet;

}

Stand\* existingStand = this->stand;

this->stand = aNewStand;

if (existingStand != NULL && existingStand != aNewStand){

existingStand->removeUser(this);

}

this->stand->addUser(this);

wasSet = true;

return wasSet;

}

// line 0 ""

string User::getId(void) const{

return this->id;

}

// line 0 ""

Stand\* User::getStand(void){

return this->stand;

}

// line 0 ""

int User::minimumNumberOfVotes() const{

//TODO Add code body to minimumNumberOfVotes

return 0;

}

// line 0 ""

int User::numberOfVotes(void) const{

int number=this->votes->size();

return number;

}

// line 0 ""

bool User::hasVotes() const{

//TODO Add code body to hasVotes

return NULL;

}

// line 0 ""

bool User::addVote(Vote\* aVote) const{

//TODO Add code body to addVote

return NULL;

}

// line 0 ""

bool User::removeVote(Vote\* aVote) const{

bool wasRemoved= false;

int index= IndexOf(this->votes->begin(), this->votes->end(), aVote);

if(index<0){

return wasRemoved;

}

//Unable to remove aVote, as it must always have a User

if (this== aVote->getUser()){

return wasRemoved;

}

wasRemoved= this->removeVoteAt(index);

return wasRemoved;

}

// line 0 ""

int User::indexOfVote(Vote\* aVote) const{

return IndexOf(this->votes->begin(), this->votes->end(), aVote);

}

// line 0 ""

bool User::addVoteAt(Vote\* aVote, const int index) const{

bool wasAdded= false;

if(addVote(aNewVote)){

int newIndex= index;

if(index < 0 ) {

newIndex = 0;

}

int size= this->numberOfVotes();

if(size>0){

this->votes->erase(this->votes->begin()+IndexOf(this->votes->begin(), this->votes->end(),aNewVote));

}

size= this->numberOfVotes();

if(newIndex > size) {

newIndex = size;

}

this->votes->insert(this->votes->begin()+ newIndex, aNewVote);

wasAdded = true;

}

return wasAdded;

}

// line 0 ""

bool User::addOrMoveVoteAt(Vote\* aVote, const int index) const{

//TODO Add code body to addOrMoveVoteAt

return NULL;

}

// line 0 ""

Vote\* User::getVote(const int index){

Vote\* aVote= (\*(this->votes))[index];

return aVote;

}

// line 0 ""

vector<Vote\*>\* User::getVotes(void){

vector<Vote\*>\* copyOfVotes = new vector<Vote\*>();

vector<Vote\*>::iterator votesBeginIterator(this->votes->begin());

vector<Vote\*>::iterator votesEndIterator(this->votes->end());

copyVector(votesBeginIterator, votesEndIterator, copyOfVotes);

return copyOfVotes;

}

bool User::hasVote(void){

return this->numberOfVotes()>0;

}

bool User::removeVoteAt(int index){

bool wasRemoved= false;

if(index<0){

return wasRemoved;

}

int size= this->numberOfVotes();

if(index> size){

return false;

}

this->votes->erase(this->votes->begin()+index);

wasRemoved= size== this->numberOfVotes()+1;

return wasRemoved;

}

bool User::addVote(Vote\* aNewVote){

bool wasAdded= false;

if (IndexOf(this->votes->begin(), this->votes->end(), aNewVote)> -1) {

return wasAdded;

}

User\* existingUser = aNewVote->getUser();

bool isNewUser= existingUser != NULL && this!=existingUser;

if(isNewUser){

aNewVote->setUser(this);

}else{

this->votes->push\_back(aNewVote);

}

wasAdded= true;

return wasAdded;

}

bool User::addOrMoveVoteAt(Vote\* aNewVote, int index){

bool wasAdded= false;

if(IndexOf(this->votes->begin(), this->votes->end(), aNewVote) > -1) {

if (index < 0) {

index = 0;

}

int size= this->numberOfVotes();

if (index > size) {

index = size - 1;

}

wasAdded= this->removeVote(aNewVote)| this->addVoteAt(aNewVote, index);

}else{

wasAdded= this->addVoteAt(aNewVote, index);

}

return wasAdded;

}

Vote\* User::addVote(const int aScore, const string aTime, Stand\* aStand, Statistics\* aStatistics){

return new Vote(aScore, aTime, aStand, this, aStatistics);

}

size\_t User::hashCode(void){

return reinterpret\_cast<size\_t>(this);

}

//------------------------

//DESTRUCTOR IMPLEMENTATION

//------------------------

User::~User(){

this->deleteAssociatedObjects();

std::for\_each(votes->begin(), votes->end(), /\*std::default\_delete<Vote>\*/ deleteElement());

votes->clear();

delete votes;

delete stand;

}

void User::deleteAssociatedObjects(void){

for(unsigned int index = votes->size(); index > 0 ; index--){

Vote\* aVote= (\*(this->votes))[index-1];

aVote->deleteAssociatedObjects();

}

Stand\* placeholderStand = this->stand;

this->stand = NULL;

placeholderStand->removeUser(this);

}

;

//%% NEW FILE Stand.cpp BEGINS HERE %%

//PLEASE DO NOT EDIT THIS CODE

//This code was generated using the UMPLE 1.29.1.4448.81a70243a modeling language

#define DEF\_\_STAND\_BODY

#if defined( PRAGMA ) && ! defined( PRAGMA\_IMPLEMENTED )

#pragma implementation <Stand.h>

#endif

//------------------------

//FILE INCLUDES

//------------------------

#include <model\_Model.h>

#include <Stand.h>

#include <Vote.h>

#include <User.h>

#include <Statistics.h>

//------------------------

//LIBRARY INCLUDES

//------------------------

using namespace std;

#include <ostream>

#include <string>

#include <vector>

//------------------------

//CONSTRUCTOR IMPLEMENTATION

//------------------------

Stand::Stand(const string aNr, const int aTotalScore){

this->nr= aNr;

this->totalScore= aTotalScore;

this->votes= new vector<Vote\*>();

this->users= new vector<User\*>();

}

Stand::Stand(Stand& other){

internalCopy(other);

}

//------------------------

//STREAM HELPER GROUP IMPLEMENTATION

//------------------------

void Stand::toOstream(ostream& os) const{

Stand\* thisptr = const\_cast<Stand\*>(this);

os << "[" << "nr:" << thisptr->getNr()<< ", " <<

"totalScore:" << thisptr->getTotalScore() << "]" << endl;

(thisptr->getVotes()!= NULL ? os << "votes:" << thisptr->getVotes() : os << "votes:" << "NULL")<< endl;

(thisptr->getUsers()!= NULL ? os << "users:" << thisptr->getUsers() : os << "users:" << "NULL");

}

ostream& operator<<(ostream& os, const Stand& dt){

dt.toOstream(os);

return os;

}

//------------------------

//PREDEFINED OPERATORS IMPLEMENTATION

//------------------------

bool operator == (Stand& Right, Stand& Left){

//if (typeid(Right) != typeid(Left)) {

// return false;

//}

if(Right.hashCode() != Left.hashCode()) {

return false;

}

if(Right.nr.compare(Left.nr) != 0){

return false;

}

if(Right.totalScore!= Left.totalScore){

return false;

}

if(!std::equal(Right.votes->begin(), Right.votes->end(), Left.votes->begin())){

return false;

}

if(!std::equal(Right.users->begin(), Right.users->end(), Left.users->begin())){

return false;

}

}

Stand& Stand::operator=(Stand& other){

internalCopy(other);

return \*this;

}

void Stand::internalCopy(Stand& other){

this->nr.assign(other.nr);

this->totalScore= other.totalScore;

this->votes = new vector<Vote\*>();

vector<Vote\*>::iterator votesBeginIterator(other.votes->begin());

vector<Vote\*>::iterator votesEndIterator(other.votes->end());

copyVector(votesBeginIterator, votesEndIterator, this->votes);

this->users = new vector<User\*>();

vector<User\*>::iterator usersBeginIterator(other.users->begin());

vector<User\*>::iterator usersEndIterator(other.users->end());

copyVector(usersBeginIterator, usersEndIterator, this->users);

}

bool Stand::setNr(const string aNewNr){

bool wasSet= false;

this->nr = aNewNr;

wasSet= true;

return wasSet;

}

bool Stand::setTotalScore(const int aNewTotalScore){

bool wasSet= false;

this->totalScore = aNewTotalScore;

wasSet= true;

return wasSet;

}

// line 0 ""

string Stand::getNr(void) const{

return this->nr;

}

// line 0 ""

int Stand::getTotalScore(void) const{

return this->totalScore;

}

// line 0 ""

int Stand::minimumNumberOfVotes() const{

//TODO Add code body to minimumNumberOfVotes

return 0;

}

// line 0 ""

int Stand::numberOfVotes(void) const{

int number=this->votes->size();

return number;

}

// line 0 ""

bool Stand::hasVotes() const{

//TODO Add code body to hasVotes

return NULL;

}

// line 0 ""

bool Stand::addVote(Vote\* aVote) const{

//TODO Add code body to addVote

return NULL;

}

// line 0 ""

bool Stand::removeVote(Vote\* aVote) const{

bool wasRemoved= false;

int index= IndexOf(this->votes->begin(), this->votes->end(), aVote);

if(index<0){

return wasRemoved;

}

//Unable to remove aVote, as it must always have a Stand

if (this== aVote->getStand()){

return wasRemoved;

}

wasRemoved= this->removeVoteAt(index);

return wasRemoved;

}

// line 0 ""

int Stand::indexOfVote(Vote\* aVote) const{

return IndexOf(this->votes->begin(), this->votes->end(), aVote);

}

// line 0 ""

bool Stand::addVoteAt(Vote\* aVote, const int index) const{

bool wasAdded= false;

if(addVote(aNewVote)){

int newIndex= index;

if(index < 0 ) {

newIndex = 0;

}

int size= this->numberOfVotes();

if(size>0){

this->votes->erase(this->votes->begin()+IndexOf(this->votes->begin(), this->votes->end(),aNewVote));

}

size= this->numberOfVotes();

if(newIndex > size) {

newIndex = size;

}

this->votes->insert(this->votes->begin()+ newIndex, aNewVote);

wasAdded = true;

}

return wasAdded;

}

// line 0 ""

bool Stand::addOrMoveVoteAt(Vote\* aVote, const int index) const{

//TODO Add code body to addOrMoveVoteAt

return NULL;

}

// line 0 ""

Vote\* Stand::getVote(const int index){

Vote\* aVote= (\*(this->votes))[index];

return aVote;

}

// line 0 ""

vector<Vote\*>\* Stand::getVotes(void){

vector<Vote\*>\* copyOfVotes = new vector<Vote\*>();

vector<Vote\*>::iterator votesBeginIterator(this->votes->begin());

vector<Vote\*>::iterator votesEndIterator(this->votes->end());

copyVector(votesBeginIterator, votesEndIterator, copyOfVotes);

return copyOfVotes;

}

// line 0 ""

int Stand::minimumNumberOfUsers() const{

//TODO Add code body to minimumNumberOfUsers

return 0;

}

// line 0 ""

int Stand::numberOfUsers(void) const{

int number=this->users->size();

return number;

}

// line 0 ""

bool Stand::hasUsers() const{

//TODO Add code body to hasUsers

return NULL;

}

// line 0 ""

bool Stand::addUser(User\* aUser) const{

//TODO Add code body to addUser

return NULL;

}

// line 0 ""

bool Stand::removeUser(User\* aUser) const{

bool wasRemoved= false;

int index= IndexOf(this->users->begin(), this->users->end(), aUser);

if(index<0){

return wasRemoved;

}

//Unable to remove aUser, as it must always have a Stand

if (this== aUser->getStand()){

return wasRemoved;

}

wasRemoved= this->removeUserAt(index);

return wasRemoved;

}

// line 0 ""

int Stand::indexOfUser(User\* aUser) const{

return IndexOf(this->users->begin(), this->users->end(), aUser);

}

// line 0 ""

bool Stand::addUserAt(User\* aUser, const int index) const{

bool wasAdded= false;

if(addUser(aNewUser)){

int newIndex= index;

if(index < 0 ) {

newIndex = 0;

}

int size= this->numberOfUsers();

if(size>0){

this->users->erase(this->users->begin()+IndexOf(this->users->begin(), this->users->end(),aNewUser));

}

size= this->numberOfUsers();

if(newIndex > size) {

newIndex = size;

}

this->users->insert(this->users->begin()+ newIndex, aNewUser);

wasAdded = true;

}

return wasAdded;

}

// line 0 ""

bool Stand::addOrMoveUserAt(User\* aUser, const int index) const{

//TODO Add code body to addOrMoveUserAt

return NULL;

}

// line 0 ""

User\* Stand::getUser(const int index){

User\* aUser= (\*(this->users))[index];

return aUser;

}

// line 0 ""

vector<User\*>\* Stand::getUsers(void){

vector<User\*>\* copyOfUsers = new vector<User\*>();

vector<User\*>::iterator usersBeginIterator(this->users->begin());

vector<User\*>::iterator usersEndIterator(this->users->end());

copyVector(usersBeginIterator, usersEndIterator, copyOfUsers);

return copyOfUsers;

}

bool Stand::hasVote(void){

return this->numberOfVotes()>0;

}

bool Stand::removeVoteAt(int index){

bool wasRemoved= false;

if(index<0){

return wasRemoved;

}

int size= this->numberOfVotes();

if(index> size){

return false;

}

this->votes->erase(this->votes->begin()+index);

wasRemoved= size== this->numberOfVotes()+1;

return wasRemoved;

}

bool Stand::addVote(Vote\* aNewVote){

bool wasAdded= false;

if (IndexOf(this->votes->begin(), this->votes->end(), aNewVote)> -1) {

return wasAdded;

}

Stand\* existingStand = aNewVote->getStand();

bool isNewStand= existingStand != NULL && this!=existingStand;

if(isNewStand){

aNewVote->setStand(this);

}else{

this->votes->push\_back(aNewVote);

}

wasAdded= true;

return wasAdded;

}

bool Stand::addOrMoveVoteAt(Vote\* aNewVote, int index){

bool wasAdded= false;

if(IndexOf(this->votes->begin(), this->votes->end(), aNewVote) > -1) {

if (index < 0) {

index = 0;

}

int size= this->numberOfVotes();

if (index > size) {

index = size - 1;

}

wasAdded= this->removeVote(aNewVote)| this->addVoteAt(aNewVote, index);

}else{

wasAdded= this->addVoteAt(aNewVote, index);

}

return wasAdded;

}

Vote\* Stand::addVote(const int aScore, const string aTime, User\* aUser, Statistics\* aStatistics){

return new Vote(aScore, aTime, this, aUser, aStatistics);

}

bool Stand::hasUser(void){

return this->numberOfUsers()>0;

}

bool Stand::removeUserAt(int index){

bool wasRemoved= false;

if(index<0){

return wasRemoved;

}

int size= this->numberOfUsers();

if(index> size){

return false;

}

this->users->erase(this->users->begin()+index);

wasRemoved= size== this->numberOfUsers()+1;

return wasRemoved;

}

bool Stand::addUser(User\* aNewUser){

bool wasAdded= false;

if (IndexOf(this->users->begin(), this->users->end(), aNewUser)> -1) {

return wasAdded;

}

Stand\* existingStand = aNewUser->getStand();

bool isNewStand= existingStand != NULL && this!=existingStand;

if(isNewStand){

aNewUser->setStand(this);

}else{

this->users->push\_back(aNewUser);

}

wasAdded= true;

return wasAdded;

}

bool Stand::addOrMoveUserAt(User\* aNewUser, int index){

bool wasAdded= false;

if(IndexOf(this->users->begin(), this->users->end(), aNewUser) > -1) {

if (index < 0) {

index = 0;

}

int size= this->numberOfUsers();

if (index > size) {

index = size - 1;

}

wasAdded= this->removeUser(aNewUser)| this->addUserAt(aNewUser, index);

}else{

wasAdded= this->addUserAt(aNewUser, index);

}

return wasAdded;

}

User\* Stand::addUser(const string aId){

return new User(aId, this);

}

size\_t Stand::hashCode(void){

return reinterpret\_cast<size\_t>(this);

}

//------------------------

//DESTRUCTOR IMPLEMENTATION

//------------------------

Stand::~Stand(){

this->deleteAssociatedObjects();

std::for\_each(votes->begin(), votes->end(), /\*std::default\_delete<Vote>\*/ deleteElement());

votes->clear();

delete votes;

std::for\_each(users->begin(), users->end(), /\*std::default\_delete<User>\*/ deleteElement());

users->clear();

delete users;

}

void Stand::deleteAssociatedObjects(void){

for(unsigned int index = votes->size(); index > 0 ; index--){

Vote\* aVote= (\*(this->votes))[index-1];

aVote->deleteAssociatedObjects();

}

for(unsigned int index = users->size(); index > 0 ; index--){

User\* aUser= (\*(this->users))[index-1];

aUser->deleteAssociatedObjects();

}

}

;

//%% NEW FILE CMakeLists.txt BEGINS HERE %%

cmake\_minimum\_required(VERSION 3.0)

project(model)

set(CMAKE\_CXX\_STANDARD 11)

include\_directories(${CMAKE\_SOURCE\_DIR})

set(SOURCE\_FILES

model\_Model.h

model\_Main.cpp

Stand.h

Stand.cpp

Vote.h

Vote.cpp

User.h

User.cpp

Jury.h

Jury.cpp

Statistics.h

Statistics.cpp)

add\_executable(model ${SOURCE\_FILES})

//%% NEW FILE Vote.cpp BEGINS HERE %%

//PLEASE DO NOT EDIT THIS CODE

//This code was generated using the UMPLE 1.29.1.4448.81a70243a modeling language

#define DEF\_\_VOTE\_BODY

#if defined( PRAGMA ) && ! defined( PRAGMA\_IMPLEMENTED )

#pragma implementation <Vote.h>

#endif

//------------------------

//FILE INCLUDES

//------------------------

#include <model\_Model.h>

#include <Vote.h>

#include <Stand.h>

#include <User.h>

#include <Statistics.h>

//------------------------

//LIBRARY INCLUDES

//------------------------

using namespace std;

#include <ostream>

//------------------------

//CONSTRUCTOR IMPLEMENTATION

//------------------------

Vote::Vote(const int aScore, const string aTime, Stand\* aStand, User\* aUser, Statistics\* aStatistics):

stand(NULL),

user(NULL),

statistics(NULL){

this->score= aScore;

this->time= aTime;

if (!setStand(aStand)){

throw "Unable to create Vote due to aStand";

}

if (!setUser(aUser)){

throw "Unable to create Vote due to aUser";

}

if (!setStatistic(aStatistic)){

throw "Unable to create Vote due to aStatistic";

}

}

Vote::Vote(Vote& other):

stand(NULL),

user(NULL),

statistics(NULL){

internalCopy(other);

}

//------------------------

//STREAM HELPER GROUP IMPLEMENTATION

//------------------------

void Vote::toOstream(ostream& os) const{

Vote\* thisptr = const\_cast<Vote\*>(this);

os << "[" << "score:" << thisptr->getScore()<< ", " <<

"time:" << thisptr->getTime() << "]" << endl;

(thisptr->getStand()!= NULL ? os << "stand:" << thisptr->getStand() : os << "stand:" << "NULL")<< endl;

(thisptr->getUser()!= NULL ? os << "user:" << thisptr->getUser() : os << "user:" << "NULL")<< endl;

(thisptr->getStatistic()!= NULL ? os << "statistics:" << thisptr->getStatistic() : os << "statistics:" << "NULL");

}

ostream& operator<<(ostream& os, const Vote& dt){

dt.toOstream(os);

return os;

}

//------------------------

//PREDEFINED OPERATORS IMPLEMENTATION

//------------------------

bool operator == (Vote& Right, Vote& Left){

//if (typeid(Right) != typeid(Left)) {

// return false;

//}

if(Right.hashCode() != Left.hashCode()) {

return false;

}

if(Right.score!= Left.score){

return false;

}

if(Right.time.compare(Left.time) != 0){

return false;

}

if(!compare(Right.stand, Left.stand, sizeof Right.stand)){

return false;

}

if(!compare(Right.user, Left.user, sizeof Right.user)){

return false;

}

if(!compare(Right.statistics, Left.statistics, sizeof Right.statistics)){

return false;

}

}

Vote& Vote::operator=(Vote& other){

internalCopy(other);

return \*this;

}

void Vote::internalCopy(Vote& other){

this->score= other.score;

this->time.assign(other.time);

copyObject(other.stand, this->stand, sizeof other.stand);

copyObject(other.user, this->user, sizeof other.user);

copyObject(other.statistics, this->statistics, sizeof other.statistics);

}

bool Vote::setScore(const int aNewScore){

bool wasSet= false;

this->score = aNewScore;

wasSet= true;

return wasSet;

}

bool Vote::setTime(const string aNewTime){

bool wasSet= false;

this->time = aNewTime;

wasSet= true;

return wasSet;

}

bool Vote::setStand(Stand\* aNewStand){

bool wasSet= false;

if (aNewStand == NULL){

return wasSet;

}

Stand\* existingStand = this->stand;

this->stand = aNewStand;

if (existingStand != NULL && existingStand != aNewStand){

existingStand->removeVote(this);

}

this->stand->addVote(this);

wasSet = true;

return wasSet;

}

bool Vote::setUser(User\* aNewUser){

bool wasSet= false;

if (aNewUser == NULL){

return wasSet;

}

User\* existingUser = this->user;

this->user = aNewUser;

if (existingUser != NULL && existingUser != aNewUser){

existingUser->removeVote(this);

}

this->user->addVote(this);

wasSet = true;

return wasSet;

}

bool Vote::setStatistic(Statistics\* aNewStatistic){

bool wasSet= false;

if (aNewStatistic == NULL){

return wasSet;

}

Statistics\* existingStatistics = this->statistics;

this->statistics = aNewStatistic;

if (existingStatistics != NULL && existingStatistics != aNewStatistic){

existingStatistics->removeVote(this);

}

this->statistics->addVote(this);

wasSet = true;

return wasSet;

}

// line 0 ""

int Vote::getScore(void) const{

return this->score;

}

// line 0 ""

string Vote::getTime(void) const{

return this->time;

}

// line 0 ""

Stand\* Vote::getStand(void){

return this->stand;

}

// line 0 ""

User\* Vote::getUser(void){

return this->user;

}

// line 0 ""

Statistics\* Vote::getStatistics(){

//TODO Add code body to getStatistics

return NULL;

}

// line 0 ""

bool Vote::setStatistics(Statistic aStatistics) const{

//TODO Add code body to setStatistics

return NULL;

}

Statistics\* Vote::getStatistic(void){

return this->statistics;

}

size\_t Vote::hashCode(void){

return reinterpret\_cast<size\_t>(this);

}

//------------------------

//DESTRUCTOR IMPLEMENTATION

//------------------------

Vote::~Vote(){

this->deleteAssociatedObjects();

delete stand;

delete user;

delete statistics;

}

void Vote::deleteAssociatedObjects(void){

Stand\* placeholderStand = this->stand;

this->stand = NULL;

placeholderStand->removeVote(this);

User\* placeholderUser = this->user;

this->user = NULL;

placeholderUser->removeVote(this);

Statistics\* placeholderStatistics = this->statistics;

this->statistics = NULL;

placeholderStatistics->removeVote(this);

}

;

//%% NEW FILE Stand.h BEGINS HERE %%

//PLEASE DO NOT EDIT THIS CODE

//This code was generated using the UMPLE 1.29.1.4448.81a70243a modeling language

#ifndef DEF\_\_STAND\_H

#define DEF\_\_STAND\_H

#ifdef PRAGMA

#pragma once

#ifndef \_MSC\_VER

#pragma interface "Stand.h"

#endif

#endif

//------------------------

//FILE INCLUDES

//------------------------

#include <model\_Model.h>

#include <Stand.h>

//------------------------

//LIBRARY INCLUDES

//------------------------

using namespace std;

#include <ostream>

#include <string>

#include <vector>

class Vote;

class User;

class Statistics;

/\*

\* Positioning

\*/

// line 2 "model.ump"

// line 38 "model.ump"

class Stand{

private:

//Stand Attributes

string nr;

int totalScore;

//Stand Associations

// line 6 "model.ump"

vector<Vote\*>\* votes;

// line 7 "model.ump"

vector<User\*>\* users;

public:

//------------------------

//CONSTRUCTOR

//------------------------

Stand(const string aNr, const int aTotalScore);

Stand(Stand& other);

//------------------------

//STREAM HELPER GROUPDECLARATION

//------------------------

friend ostream& operator<<(ostream& os, const Stand& dt);

//------------------------

//PREDEFINED OPERATORS

//------------------------

friend bool operator == (Stand& Right, Stand& Left);

friend bool operator != (Stand& Right, Stand& Left){

return !( Right == Left);

}

bool operator == (const Stand& Right) const{

return this == &Right;

}

bool operator != (const Stand& Right) const{

return this != &Right;

}

Stand& operator=(Stand& other);

void internalCopy(Stand& other);

bool setNr(const string aNewNr);

bool setTotalScore(const int aNewTotalScore);

// line 0 ""

string getNr(void) const;

// line 0 ""

int getTotalScore(void) const;

// line 0 ""

int minimumNumberOfVotes() const;

// line 0 ""

int numberOfVotes(void) const;

// line 0 ""

bool hasVotes() const;

// line 0 ""

bool addVote(Vote\* aVote) const;

// line 0 ""

bool removeVote(Vote\* aVote) const;

// line 0 ""

int indexOfVote(Vote\* aVote) const;

// line 0 ""

bool addVoteAt(Vote\* aVote, const int index) const;

// line 0 ""

bool addOrMoveVoteAt(Vote\* aVote, const int index) const;

// line 0 ""

Vote\* getVote(const int index);

// line 0 ""

vector<Vote\*>\* getVotes(void);

// line 0 ""

int minimumNumberOfUsers() const;

// line 0 ""

int numberOfUsers(void) const;

// line 0 ""

bool hasUsers() const;

// line 0 ""

bool addUser(User\* aUser) const;

// line 0 ""

bool removeUser(User\* aUser) const;

// line 0 ""

int indexOfUser(User\* aUser) const;

// line 0 ""

bool addUserAt(User\* aUser, const int index) const;

// line 0 ""

bool addOrMoveUserAt(User\* aUser, const int index) const;

// line 0 ""

User\* getUser(const int index);

// line 0 ""

vector<User\*>\* getUsers(void);

bool hasVote(void);

bool removeVoteAt(int index);

bool addVote(Vote\* aNewVote);

bool addOrMoveVoteAt(Vote\* aNewVote, int index);

Vote\* addVote(const int aScore, const string aTime, User\* aUser, Statistics\* aStatistics);

bool hasUser(void);

bool removeUserAt(int index);

bool addUser(User\* aNewUser);

bool addOrMoveUserAt(User\* aNewUser, int index);

User\* addUser(const string aId);

virtual size\_t hashCode(void);

//------------------------

//DESTRUCTOR

//------------------------

virtual ~Stand();

void deleteAssociatedObjects(void);

protected:

//------------------------

//STREAM HELPER GROUPDECLARATION

//------------------------

virtual void toOstream(ostream& os) const;

};

//------------------------

//GNU HASH FUNCTION USE

//------------------------

#ifdef \_\_GNUC\_\_

using namespace \_\_gnu\_cxx;

namespace \_\_gnu\_cxx{

template<> struct hash<Stand\*>{

size\_t operator()(Stand\* ptr ) const {

return ptr->hashCode();

}

};

}

#include <ext/hash\_map>

#else

#include <hash\_map>

#endif

#endif

//%% NEW FILE User.h BEGINS HERE %%

//PLEASE DO NOT EDIT THIS CODE

//This code was generated using the UMPLE 1.29.1.4448.81a70243a modeling language

#ifndef DEF\_\_USER\_H

#define DEF\_\_USER\_H

#ifdef PRAGMA

#pragma once

#ifndef \_MSC\_VER

#pragma interface "User.h"

#endif

#endif

//------------------------

//FILE INCLUDES

//------------------------

#include <model\_Model.h>

#include <User.h>

//------------------------

//LIBRARY INCLUDES

//------------------------

using namespace std;

#include <ostream>

#include <vector>

class Stand;

class Vote;

class Statistics;

// line 16 "model.ump"

// line 51 "model.ump"

class User{

private:

//User Attributes

string id;

//User Associations

// line 19 "model.ump"

vector<Vote\*>\* votes;

// line 7 "model.ump"

Stand\* stand;

public:

//------------------------

//CONSTRUCTOR

//------------------------

User(const string aId, Stand\* aStand);

User(User& other);

//------------------------

//STREAM HELPER GROUPDECLARATION

//------------------------

friend ostream& operator<<(ostream& os, const User& dt);

//------------------------

//PREDEFINED OPERATORS

//------------------------

friend bool operator == (User& Right, User& Left);

friend bool operator != (User& Right, User& Left){

return !( Right == Left);

}

bool operator == (const User& Right) const{

return this == &Right;

}

bool operator != (const User& Right) const{

return this != &Right;

}

User& operator=(User& other);

void internalCopy(User& other);

bool setId(const string aNewId);

bool setStand(Stand\* aNewStand);

inline unsigned int minimumNumberOfStands(void){

return 1;

}

// line 0 ""

string getId(void) const;

// line 0 ""

Stand\* getStand(void);

// line 0 ""

int minimumNumberOfVotes() const;

// line 0 ""

int numberOfVotes(void) const;

// line 0 ""

bool hasVotes() const;

// line 0 ""

bool addVote(Vote\* aVote) const;

// line 0 ""

bool removeVote(Vote\* aVote) const;

// line 0 ""

int indexOfVote(Vote\* aVote) const;

// line 0 ""

bool addVoteAt(Vote\* aVote, const int index) const;

// line 0 ""

bool addOrMoveVoteAt(Vote\* aVote, const int index) const;

// line 0 ""

Vote\* getVote(const int index);

// line 0 ""

vector<Vote\*>\* getVotes(void);

bool hasVote(void);

bool removeVoteAt(int index);

bool addVote(Vote\* aNewVote);

bool addOrMoveVoteAt(Vote\* aNewVote, int index);

Vote\* addVote(const int aScore, const string aTime, Stand\* aStand, Statistics\* aStatistics);

virtual size\_t hashCode(void);

//------------------------

//DESTRUCTOR

//------------------------

virtual ~User();

void deleteAssociatedObjects(void);

protected:

//------------------------

//STREAM HELPER GROUPDECLARATION

//------------------------

virtual void toOstream(ostream& os) const;

};

//------------------------

//GNU HASH FUNCTION USE

//------------------------

#ifdef \_\_GNUC\_\_

using namespace \_\_gnu\_cxx;

namespace \_\_gnu\_cxx{

template<> struct hash<User\*>{

size\_t operator()(User\* ptr ) const {

return ptr->hashCode();

}

};

}

#include <ext/hash\_map>

#else

#include <hash\_map>

#endif

#endif

//%% NEW FILE Vote.h BEGINS HERE %%

//PLEASE DO NOT EDIT THIS CODE

//This code was generated using the UMPLE 1.29.1.4448.81a70243a modeling language

#ifndef DEF\_\_VOTE\_H

#define DEF\_\_VOTE\_H

#ifdef PRAGMA

#pragma once

#ifndef \_MSC\_VER

#pragma interface "Vote.h"

#endif

#endif

//------------------------

//FILE INCLUDES

//------------------------

#include <model\_Model.h>

#include <Vote.h>

//------------------------

//LIBRARY INCLUDES

//------------------------

using namespace std;

#include <ostream>

class Stand;

class User;

class Statistics;

// line 10 "model.ump"

// line 46 "model.ump"

class Vote{

private:

//Vote Attributes

int score;

string time;

//Vote Associations

// line 6 "model.ump"

Stand\* stand;

// line 19 "model.ump"

User\* user;

// line 32 "model.ump"

Statistics\* statistics;

public:

//------------------------

//CONSTRUCTOR

//------------------------

Vote(const int aScore, const string aTime, Stand\* aStand, User\* aUser, Statistics\* aStatistics);

Vote(Vote& other);

//------------------------

//STREAM HELPER GROUPDECLARATION

//------------------------

friend ostream& operator<<(ostream& os, const Vote& dt);

//------------------------

//PREDEFINED OPERATORS

//------------------------

friend bool operator == (Vote& Right, Vote& Left);

friend bool operator != (Vote& Right, Vote& Left){

return !( Right == Left);

}

bool operator == (const Vote& Right) const{

return this == &Right;

}

bool operator != (const Vote& Right) const{

return this != &Right;

}

Vote& operator=(Vote& other);

void internalCopy(Vote& other);

bool setScore(const int aNewScore);

bool setTime(const string aNewTime);

bool setStand(Stand\* aNewStand);

inline unsigned int minimumNumberOfStands(void){

return 1;

}

bool setUser(User\* aNewUser);

inline unsigned int minimumNumberOfUsers(void){

return 1;

}

bool setStatistic(Statistics\* aNewStatistic);

inline unsigned int minimumNumberOfStatistics(void){

return 1;

}

// line 0 ""

int getScore(void) const;

// line 0 ""

string getTime(void) const;

// line 0 ""

Stand\* getStand(void);

// line 0 ""

User\* getUser(void);

// line 0 ""

Statistics\* getStatistics();

// line 0 ""

bool setStatistics(Statistic aStatistics) const;

Statistics\* getStatistic(void);

virtual size\_t hashCode(void);

//------------------------

//DESTRUCTOR

//------------------------

virtual ~Vote();

void deleteAssociatedObjects(void);

protected:

//------------------------

//STREAM HELPER GROUPDECLARATION

//------------------------

virtual void toOstream(ostream& os) const;

};

//------------------------

//GNU HASH FUNCTION USE

//------------------------

#ifdef \_\_GNUC\_\_

using namespace \_\_gnu\_cxx;

namespace \_\_gnu\_cxx{

template<> struct hash<Vote\*>{

size\_t operator()(Vote\* ptr ) const {

return ptr->hashCode();

}

};

}

#include <ext/hash\_map>

#else

#include <hash\_map>

#endif

#endif

//%% NEW FILE Statistics.h BEGINS HERE %%

//PLEASE DO NOT EDIT THIS CODE

//This code was generated using the UMPLE 1.29.1.4448.81a70243a modeling language

#ifndef DEF\_\_STATISTICS\_H

#define DEF\_\_STATISTICS\_H

#ifdef PRAGMA

#pragma once

#ifndef \_MSC\_VER

#pragma interface "Statistics.h"

#endif

#endif

//------------------------

//FILE INCLUDES

//------------------------

#include <model\_Model.h>

#include <Statistics.h>

//------------------------

//LIBRARY INCLUDES

//------------------------

using namespace std;

#include <ostream>

#include <vector>

class Jury;

class Vote;

class Stand;

class User;

// line 28 "model.ump"

// line 64 "model.ump"

class Statistics{

private:

//Statistics Attributes

string password;

//Statistics Associations

// line 32 "model.ump"

vector<Vote\*>\* votes;

// line 25 "model.ump"

vector<Jury\*>\* juries;

public:

//------------------------

//CONSTRUCTOR

//------------------------

Statistics(const string aPassword);

Statistics(Statistics& other);

//------------------------

//STREAM HELPER GROUPDECLARATION

//------------------------

friend ostream& operator<<(ostream& os, const Statistics& dt);

//------------------------

//PREDEFINED OPERATORS

//------------------------

friend bool operator == (Statistics& Right, Statistics& Left);

friend bool operator != (Statistics& Right, Statistics& Left){

return !( Right == Left);

}

bool operator == (const Statistics& Right) const{

return this == &Right;

}

bool operator != (const Statistics& Right) const{

return this != &Right;

}

Statistics& operator=(Statistics& other);

void internalCopy(Statistics& other);

bool setPassword(const string aNewPassword);

// line 0 ""

string getPassword(void) const;

// line 0 ""

int minimumNumberOfJuries() const;

// line 0 ""

int numberOfJuries(void) const;

// line 0 ""

bool hasJuries() const;

// line 0 ""

bool addJury(Jury\* aJury) const;

// line 0 ""

bool removeJury(Jury\* aJury) const;

// line 0 ""

int indexOfJury(Jury\* aJury) const;

// line 0 ""

bool addJuryAt(Jury\* aJury, const int index) const;

// line 0 ""

bool addOrMoveJuryAt(Jury\* aJury, const int index) const;

// line 0 ""

Jury\* getJury(const int index);

// line 0 ""

vector<Jury\*>\* getJuries(void);

// line 0 ""

int minimumNumberOfVotes() const;

// line 0 ""

int numberOfVotes(void) const;

// line 0 ""

bool hasVotes() const;

// line 0 ""

bool addVote(Vote\* aVote) const;

// line 0 ""

bool removeVote(Vote\* aVote) const;

// line 0 ""

int indexOfVote(Vote\* aVote) const;

// line 0 ""

bool addVoteAt(Vote\* aVote, const int index) const;

// line 0 ""

bool addOrMoveVoteAt(Vote\* aVote, const int index) const;

// line 0 ""

Vote\* getVote(const int index);

// line 0 ""

vector<Vote\*>\* getVotes(void);

bool hasVote(void);

bool removeVoteAt(int index);

bool addVote(Vote\* aNewVote);

bool addOrMoveVoteAt(Vote\* aNewVote, int index);

Vote\* addVote(const int aScore, const string aTime, Stand\* aStand, User\* aUser);

bool hasJury(void);

bool removeJuryAt(int index);

bool addJury(Jury\* aNewJury);

bool addOrMoveJuryAt(Jury\* aNewJury, int index);

Jury\* addJury(const string aId, Stand\* aStand);

virtual size\_t hashCode(void);

//------------------------

//DESTRUCTOR

//------------------------

virtual ~Statistics();

void deleteAssociatedObjects(void);

protected:

//------------------------

//STREAM HELPER GROUPDECLARATION

//------------------------

virtual void toOstream(ostream& os) const;

};

//------------------------

//GNU HASH FUNCTION USE

//------------------------

#ifdef \_\_GNUC\_\_

using namespace \_\_gnu\_cxx;

namespace \_\_gnu\_cxx{

template<> struct hash<Statistics\*>{

size\_t operator()(Statistics\* ptr ) const {

return ptr->hashCode();

}

};

}

#include <ext/hash\_map>

#else

#include <hash\_map>

#endif

#endif

//%% NEW FILE Statistics.cpp BEGINS HERE %%

//PLEASE DO NOT EDIT THIS CODE

//This code was generated using the UMPLE 1.29.1.4448.81a70243a modeling language

#define DEF\_\_STATISTICS\_BODY

#if defined( PRAGMA ) && ! defined( PRAGMA\_IMPLEMENTED )

#pragma implementation <Statistics.h>

#endif

//------------------------

//FILE INCLUDES

//------------------------

#include <model\_Model.h>

#include <Statistics.h>

#include <Jury.h>

#include <Vote.h>

#include <Stand.h>

#include <User.h>

//------------------------

//LIBRARY INCLUDES

//------------------------

using namespace std;

#include <ostream>

#include <vector>

//------------------------

//CONSTRUCTOR IMPLEMENTATION

//------------------------

Statistics::Statistics(const string aPassword){

this->password= aPassword;

this->votes= new vector<Vote\*>();

this->juries= new vector<Jury\*>();

}

Statistics::Statistics(Statistics& other){

internalCopy(other);

}

//------------------------

//STREAM HELPER GROUP IMPLEMENTATION

//------------------------

void Statistics::toOstream(ostream& os) const{

Statistics\* thisptr = const\_cast<Statistics\*>(this);

os << "[" << "password:" << thisptr->getPassword() << "]" << endl;

(thisptr->getVotes()!= NULL ? os << "votes:" << thisptr->getVotes() : os << "votes:" << "NULL")<< endl;

(thisptr->getJuries()!= NULL ? os << "juries:" << thisptr->getJuries() : os << "juries:" << "NULL");

}

ostream& operator<<(ostream& os, const Statistics& dt){

dt.toOstream(os);

return os;

}

//------------------------

//PREDEFINED OPERATORS IMPLEMENTATION

//------------------------

bool operator == (Statistics& Right, Statistics& Left){

//if (typeid(Right) != typeid(Left)) {

// return false;

//}

if(Right.hashCode() != Left.hashCode()) {

return false;

}

if(Right.password.compare(Left.password) != 0){

return false;

}

if(!std::equal(Right.votes->begin(), Right.votes->end(), Left.votes->begin())){

return false;

}

if(!std::equal(Right.juries->begin(), Right.juries->end(), Left.juries->begin())){

return false;

}

}

Statistics& Statistics::operator=(Statistics& other){

internalCopy(other);

return \*this;

}

void Statistics::internalCopy(Statistics& other){

this->password.assign(other.password);

this->votes = new vector<Vote\*>();

vector<Vote\*>::iterator votesBeginIterator(other.votes->begin());

vector<Vote\*>::iterator votesEndIterator(other.votes->end());

copyVector(votesBeginIterator, votesEndIterator, this->votes);

this->juries = new vector<Jury\*>();

vector<Jury\*>::iterator juriesBeginIterator(other.juries->begin());

vector<Jury\*>::iterator juriesEndIterator(other.juries->end());

copyVector(juriesBeginIterator, juriesEndIterator, this->juries);

}

bool Statistics::setPassword(const string aNewPassword){

bool wasSet= false;

this->password = aNewPassword;

wasSet= true;

return wasSet;

}

// line 0 ""

string Statistics::getPassword(void) const{

return this->password;

}

// line 0 ""

int Statistics::minimumNumberOfJuries() const{

//TODO Add code body to minimumNumberOfJuries

return 0;

}

// line 0 ""

int Statistics::numberOfJuries(void) const{

int number=this->juries->size();

return number;

}

// line 0 ""

bool Statistics::hasJuries() const{

//TODO Add code body to hasJuries

return NULL;

}

// line 0 ""

bool Statistics::addJury(Jury\* aJury) const{

//TODO Add code body to addJury

return NULL;

}

// line 0 ""

bool Statistics::removeJury(Jury\* aJury) const{

bool wasRemoved= false;

int index= IndexOf(this->juries->begin(), this->juries->end(), aJury);

if(index<0){

return wasRemoved;

}

//Unable to remove aJury, as it must always have a Statistics

if (this== aJury->getStatistic()){

return wasRemoved;

}

wasRemoved= this->removeJuryAt(index);

return wasRemoved;

}

// line 0 ""

int Statistics::indexOfJury(Jury\* aJury) const{

return IndexOf(this->juries->begin(), this->juries->end(), aJury);

}

// line 0 ""

bool Statistics::addJuryAt(Jury\* aJury, const int index) const{

bool wasAdded= false;

if(addJury(aNewJury)){

int newIndex= index;

if(index < 0 ) {

newIndex = 0;

}

int size= this->numberOfJuries();

if(size>0){

this->juries->erase(this->juries->begin()+IndexOf(this->juries->begin(), this->juries->end(),aNewJury));

}

size= this->numberOfJuries();

if(newIndex > size) {

newIndex = size;

}

this->juries->insert(this->juries->begin()+ newIndex, aNewJury);

wasAdded = true;

}

return wasAdded;

}

// line 0 ""

bool Statistics::addOrMoveJuryAt(Jury\* aJury, const int index) const{

//TODO Add code body to addOrMoveJuryAt

return NULL;

}

// line 0 ""

Jury\* Statistics::getJury(const int index){

Jury\* aJury= (\*(this->juries))[index];

return aJury;

}

// line 0 ""

vector<Jury\*>\* Statistics::getJuries(void){

vector<Jury\*>\* copyOfJuries = new vector<Jury\*>();

vector<Jury\*>::iterator juriesBeginIterator(this->juries->begin());

vector<Jury\*>::iterator juriesEndIterator(this->juries->end());

copyVector(juriesBeginIterator, juriesEndIterator, copyOfJuries);

return copyOfJuries;

}

// line 0 ""

int Statistics::minimumNumberOfVotes() const{

//TODO Add code body to minimumNumberOfVotes

return 0;

}

// line 0 ""

int Statistics::numberOfVotes(void) const{

int number=this->votes->size();

return number;

}

// line 0 ""

bool Statistics::hasVotes() const{

//TODO Add code body to hasVotes

return NULL;

}

// line 0 ""

bool Statistics::addVote(Vote\* aVote) const{

//TODO Add code body to addVote

return NULL;

}

// line 0 ""

bool Statistics::removeVote(Vote\* aVote) const{

bool wasRemoved= false;

int index= IndexOf(this->votes->begin(), this->votes->end(), aVote);

if(index<0){

return wasRemoved;

}

//Unable to remove aVote, as it must always have a Statistics

if (this== aVote->getStatistic()){

return wasRemoved;

}

wasRemoved= this->removeVoteAt(index);

return wasRemoved;

}

// line 0 ""

int Statistics::indexOfVote(Vote\* aVote) const{

return IndexOf(this->votes->begin(), this->votes->end(), aVote);

}

// line 0 ""

bool Statistics::addVoteAt(Vote\* aVote, const int index) const{

bool wasAdded= false;

if(addVote(aNewVote)){

int newIndex= index;

if(index < 0 ) {

newIndex = 0;

}

int size= this->numberOfVotes();

if(size>0){

this->votes->erase(this->votes->begin()+IndexOf(this->votes->begin(), this->votes->end(),aNewVote));

}

size= this->numberOfVotes();

if(newIndex > size) {

newIndex = size;

}

this->votes->insert(this->votes->begin()+ newIndex, aNewVote);

wasAdded = true;

}

return wasAdded;

}

// line 0 ""

bool Statistics::addOrMoveVoteAt(Vote\* aVote, const int index) const{

//TODO Add code body to addOrMoveVoteAt

return NULL;

}

// line 0 ""

Vote\* Statistics::getVote(const int index){

Vote\* aVote= (\*(this->votes))[index];

return aVote;

}

// line 0 ""

vector<Vote\*>\* Statistics::getVotes(void){

vector<Vote\*>\* copyOfVotes = new vector<Vote\*>();

vector<Vote\*>::iterator votesBeginIterator(this->votes->begin());

vector<Vote\*>::iterator votesEndIterator(this->votes->end());

copyVector(votesBeginIterator, votesEndIterator, copyOfVotes);

return copyOfVotes;

}

bool Statistics::hasVote(void){

return this->numberOfVotes()>0;

}

bool Statistics::removeVoteAt(int index){

bool wasRemoved= false;

if(index<0){

return wasRemoved;

}

int size= this->numberOfVotes();

if(index> size){

return false;

}

this->votes->erase(this->votes->begin()+index);

wasRemoved= size== this->numberOfVotes()+1;

return wasRemoved;

}

bool Statistics::addVote(Vote\* aNewVote){

bool wasAdded= false;

if (IndexOf(this->votes->begin(), this->votes->end(), aNewVote)> -1) {

return wasAdded;

}

Statistics\* existingStatistic = aNewVote->getStatistic();

bool isNewStatistic= existingStatistic != NULL && this!=existingStatistic;

if(isNewStatistic){

aNewVote->setStatistic(this);

}else{

this->votes->push\_back(aNewVote);

}

wasAdded= true;

return wasAdded;

}

bool Statistics::addOrMoveVoteAt(Vote\* aNewVote, int index){

bool wasAdded= false;

if(IndexOf(this->votes->begin(), this->votes->end(), aNewVote) > -1) {

if (index < 0) {

index = 0;

}

int size= this->numberOfVotes();

if (index > size) {

index = size - 1;

}

wasAdded= this->removeVote(aNewVote)| this->addVoteAt(aNewVote, index);

}else{

wasAdded= this->addVoteAt(aNewVote, index);

}

return wasAdded;

}

Vote\* Statistics::addVote(const int aScore, const string aTime, Stand\* aStand, User\* aUser){

return new Vote(aScore, aTime, aStand, aUser, this);

}

bool Statistics::hasJury(void){

return this->numberOfJuries()>0;

}

bool Statistics::removeJuryAt(int index){

bool wasRemoved= false;

if(index<0){

return wasRemoved;

}

int size= this->numberOfJuries();

if(index> size){

return false;

}

this->juries->erase(this->juries->begin()+index);

wasRemoved= size== this->numberOfJuries()+1;

return wasRemoved;

}

bool Statistics::addJury(Jury\* aNewJury){

bool wasAdded= false;

if (IndexOf(this->juries->begin(), this->juries->end(), aNewJury)> -1) {

return wasAdded;

}

Statistics\* existingStatistic = aNewJury->getStatistic();

bool isNewStatistic= existingStatistic != NULL && this!=existingStatistic;

if(isNewStatistic){

aNewJury->setStatistic(this);

}else{

this->juries->push\_back(aNewJury);

}

wasAdded= true;

return wasAdded;

}

bool Statistics::addOrMoveJuryAt(Jury\* aNewJury, int index){

bool wasAdded= false;

if(IndexOf(this->juries->begin(), this->juries->end(), aNewJury) > -1) {

if (index < 0) {

index = 0;

}

int size= this->numberOfJuries();

if (index > size) {

index = size - 1;

}

wasAdded= this->removeJury(aNewJury)| this->addJuryAt(aNewJury, index);

}else{

wasAdded= this->addJuryAt(aNewJury, index);

}

return wasAdded;

}

Jury\* Statistics::addJury(const string aId, Stand\* aStand){

return new Jury(aId, aStand, this);

}

size\_t Statistics::hashCode(void){

return reinterpret\_cast<size\_t>(this);

}

//------------------------

//DESTRUCTOR IMPLEMENTATION

//------------------------

Statistics::~Statistics(){

this->deleteAssociatedObjects();

std::for\_each(votes->begin(), votes->end(), /\*std::default\_delete<Vote>\*/ deleteElement());

votes->clear();

delete votes;

std::for\_each(juries->begin(), juries->end(), /\*std::default\_delete<Jury>\*/ deleteElement());

juries->clear();

delete juries;

}

void Statistics::deleteAssociatedObjects(void){

for(unsigned int index = votes->size(); index > 0 ; index--){

Vote\* aVote= (\*(this->votes))[index-1];

aVote->deleteAssociatedObjects();

}

for(unsigned int index = juries->size(); index > 0 ; index--){

Jury\* aJury= (\*(this->juries))[index-1];

aJury->deleteAssociatedObjects();

}

}

;

//%% NEW FILE Jury.cpp BEGINS HERE %%

//PLEASE DO NOT EDIT THIS CODE

//This code was generated using the UMPLE 1.29.1.4448.81a70243a modeling language

#define DEF\_\_JURY\_BODY

#if defined( PRAGMA ) && ! defined( PRAGMA\_IMPLEMENTED )

#pragma implementation <Jury.h>

#endif

//------------------------

//FILE INCLUDES

//------------------------

#include <model\_Model.h>

#include <Jury.h>

#include <User.h>

#include <Statistics.h>

//------------------------

//LIBRARY INCLUDES

//------------------------

using namespace std;

#include <ostream>

//------------------------

//CONSTRUCTOR IMPLEMENTATION

//------------------------

Jury::Jury(const string aId, Stand\* aStand, Statistics\* aStatistics):

statistics(NULL),

User(aId, aStand){

if (!setStatistic(aStatistic)){

throw "Unable to create Jury due to aStatistic";

}

}

Jury::Jury(Jury& other):

statistics(NULL),

User(other){

internalCopy(other);

}

//------------------------

//STREAM HELPER GROUP IMPLEMENTATION

//------------------------

void Jury::toOstream(ostream& os) const{

User::toOstream(os);

Jury\* thisptr = const\_cast<Jury\*>(this);

(thisptr->getStatistic()!= NULL ? os << "statistics:" << thisptr->getStatistic() : os << "statistics:" << "NULL");

}

ostream& operator<<(ostream& os, const Jury& dt){

dt.toOstream(os);

return os;

}

//------------------------

//PREDEFINED OPERATORS IMPLEMENTATION

//------------------------

bool operator == (Jury& Right, Jury& Left){

//if (typeid(Right) != typeid(Left)) {

// return false;

//}

if(Right.hashCode() != Left.hashCode()) {

return false;

}

if(!compare(Right.statistics, Left.statistics, sizeof Right.statistics)){

return false;

}

}

Jury& Jury::operator=(Jury& other){

internalCopy(other);

return \*this;

}

void Jury::internalCopy(Jury& other){

copyObject(other.statistics, this->statistics, sizeof other.statistics);

}

bool Jury::setStatistic(Statistics\* aNewStatistic){

bool wasSet= false;

if (aNewStatistic == NULL){

return wasSet;

}

Statistics\* existingStatistics = this->statistics;

this->statistics = aNewStatistic;

if (existingStatistics != NULL && existingStatistics != aNewStatistic){

existingStatistics->removeJury(this);

}

this->statistics->addJury(this);

wasSet = true;

return wasSet;

}

// line 0 ""

Statistics\* Jury::getStatistics(){

//TODO Add code body to getStatistics

return NULL;

}

// line 0 ""

bool Jury::setStatistics(Statistic aStatistics) const{

//TODO Add code body to setStatistics

return NULL;

}

Statistics\* Jury::getStatistic(void){

return this->statistics;

}

size\_t Jury::hashCode(void){

return reinterpret\_cast<size\_t>(this);

}

//------------------------

//DESTRUCTOR IMPLEMENTATION

//------------------------

Jury::~Jury(){

//User::~User();

this->deleteAssociatedObjects();

delete statistics;

}

void Jury::deleteAssociatedObjects(void){

Statistics\* placeholderStatistics = this->statistics;

this->statistics = NULL;

placeholderStatistics->removeJury(this);

}

;

Kode fra tilstandsmodellen:

Java:

//%% NEW FILE Vote BEGINS HERE %%

/\*PLEASE DO NOT EDIT THIS CODE\*/

/\*This code was generated using the UMPLE 1.29.1.4448.81a70243a modeling language!\*/

// line 2 "model.ump"

// line 32 "model.ump"

public class Vote

{

//------------------------

// MEMBER VARIABLES

//------------------------

//Vote State Machines

public enum State { noVote, voting, voted }

private State state;

//------------------------

// CONSTRUCTOR

//------------------------

public Vote()

{

setState(State.noVote);

}

//------------------------

// INTERFACE

//------------------------

public String getStateFullName()

{

String answer = state.toString();

return answer;

}

public State getState()

{

return state;

}

public boolean newVote()

{

boolean wasEventProcessed = false;

State aState = state;

switch (aState)

{

case noVote:

setState(State.voting);

wasEventProcessed = true;

break;

default:

// Other states do respond to this event

}

return wasEventProcessed;

}

public boolean votingDone()

{

boolean wasEventProcessed = false;

State aState = state;

switch (aState)

{

case voting:

setState(State.voted);

wasEventProcessed = true;

break;

default:

// Other states do respond to this event

}

return wasEventProcessed;

}

public boolean reVote()

{

boolean wasEventProcessed = false;

State aState = state;

switch (aState)

{

case voted:

setState(State.voting);

wasEventProcessed = true;

break;

default:

// Other states do respond to this event

}

return wasEventProcessed;

}

private void setState(State aState)

{

state = aState;

}

public void delete()

{}

}

//%% NEW FILE Stats BEGINS HERE %%

/\*PLEASE DO NOT EDIT THIS CODE\*/

/\*This code was generated using the UMPLE 1.29.1.4448.81a70243a modeling language!\*/

// line 16 "model.ump"

// line 27 "model.ump"

public class Stats

{

//------------------------

// MEMBER VARIABLES

//------------------------

//Stats State Machines

public enum State { loggInn, statistikk }

private State state;

//------------------------

// CONSTRUCTOR

//------------------------

public Stats()

{

setState(State.loggInn);

}

//------------------------

// INTERFACE

//------------------------

public String getStateFullName()

{

String answer = state.toString();

return answer;

}

public State getState()

{

return state;

}

public boolean skrivInnPassord()

{

boolean wasEventProcessed = false;

State aState = state;

switch (aState)

{

case loggInn:

setState(State.statistikk);

wasEventProcessed = true;

break;

default:

// Other states do respond to this event

}

return wasEventProcessed;

}

public boolean feilPassord()

{

boolean wasEventProcessed = false;

State aState = state;

switch (aState)

{

case loggInn:

setState(State.loggInn);

wasEventProcessed = true;

break;

default:

// Other states do respond to this event

}

return wasEventProcessed;

}

private void setState(State aState)

{

state = aState;

}

public void delete()

{}

}

PHP:

//%% NEW FILE Vote BEGINS HERE %%

/\*PLEASE DO NOT EDIT THIS CODE\*/

/\*This code was generated using the UMPLE 1.29.1.4448.81a70243a modeling language!\*/

class Vote

{

//------------------------

// MEMBER VARIABLES

//------------------------

//Vote State Machines

private static $StateNoVote = 1;

private static $StateVoting = 2;

private static $StateVoted = 3;

private $state;

//------------------------

// CONSTRUCTOR

//------------------------

public function \_\_construct()

{

$this->setState(self::$StateNoVote);

}

//------------------------

// INTERFACE

//------------------------

public function getStateFullName()

{

$answer = $this->getState();

return $answer;

}

public function getState()

{

if ($this->state == self::$StateNoVote) { return "StateNoVote"; }

elseif ($this->state == self::$StateVoting) { return "StateVoting"; }

elseif ($this->state == self::$StateVoted) { return "StateVoted"; }

return null;

}

public function newVote()

{

$wasEventProcessed = false;

$aState = $this->state;

if ($aState == self::$StateNoVote)

{

$this->setState(self::$StateVoting);

$wasEventProcessed = true;

}

return $wasEventProcessed;

}

public function votingDone()

{

$wasEventProcessed = false;

$aState = $this->state;

if ($aState == self::$StateVoting)

{

$this->setState(self::$StateVoted);

$wasEventProcessed = true;

}

return $wasEventProcessed;

}

public function reVote()

{

$wasEventProcessed = false;

$aState = $this->state;

if ($aState == self::$StateVoted)

{

$this->setState(self::$StateVoting);

$wasEventProcessed = true;

}

return $wasEventProcessed;

}

private function setState($aState)

{

$this->state = $aState;

}

public function equals($compareTo)

{

return $this == $compareTo;

}

public function delete()

{}

}

//%% NEW FILE Statistics BEGINS HERE %%

/\*PLEASE DO NOT EDIT THIS CODE\*/

/\*This code was generated using the UMPLE 1.29.1.4448.81a70243a modeling language!\*/

class Statistics

{

//------------------------

// MEMBER VARIABLES

//------------------------

//Statistics State Machines

private static $StateLoggInn = 1;

private static $StateStatistikk = 2;

private $state;

//------------------------

// CONSTRUCTOR

//------------------------

public function \_\_construct()

{

$this->setState(self::$StateLoggInn);

}

//------------------------

// INTERFACE

//------------------------

public function getStateFullName()

{

$answer = $this->getState();

return $answer;

}

public function getState()

{

if ($this->state == self::$StateLoggInn) { return "StateLoggInn"; }

elseif ($this->state == self::$StateStatistikk) { return "StateStatistikk"; }

return null;

}

public function skrivInnPassord()

{

$wasEventProcessed = false;

$aState = $this->state;

if ($aState == self::$StateLoggInn)

{

$this->setState(self::$StateStatistikk);

$wasEventProcessed = true;

}

return $wasEventProcessed;

}

public function feilPassord()

{

$wasEventProcessed = false;

$aState = $this->state;

if ($aState == self::$StateLoggInn)

{

$this->setState(self::$StateLoggInn);

$wasEventProcessed = true;

}

return $wasEventProcessed;

}

private function setState($aState)

{

$this->state = $aState;

}

public function equals($compareTo)

{

return $this == $compareTo;

}

public function delete()

{}

}