Objetinio programų projektavimo užduotis

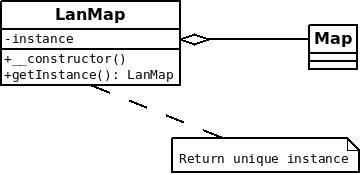
# Pavadinimas. Video žaidimų renginio valdymo programinės įrangos tobulinimas.

# Užduotis. Patobulinti atviro kodo produktą pavadinimus „Lanager“, skirtą valdyti ir administruoti video žaidimų renginius (Lan Party) . PĮ turi turėti papildomus funkcionalumus: komandų, komandų kapitonų administravimas. PĮ administratorių valdymas.Turnyrų, turnyrų lentelių valdymas pasitelkiant Challonge API sistemą (http://api.challonge.com/v1). Išsankstinis komandų, žaidėjų registravimosi modulis. Renginio vietų rezervavimas plane.

# Sistemos paskirtis ir tikslas. Administruoti Lan Party organizavimą, susisteminti visas teikiamas renginio paslaugas per PĮ. Pateikti visą informaciją apie renginį dalyviams, palengvinti dalyvių bendravimą tarpusavyje.

# Realizuojami šablonai:

* 1. Singleton šablonas, skirtas LanMap klasei. Pagrindimas: visame renginyje gali būti tik vienas interaktyvus žemėlapis, apibūdinantis renginį. Žemėlapio naudojimas bus dažnas visame projekte – naudojamas registracijose, žaidėjo padėties nusakyme.



<?php namespace Zeropingheroes\Lanager\AreaMaps;

use Zeropingheroes\Lanager\BaseModel;

use Zeropingheroes\Lanager\AreaMaps\Map;

class LanMap extends BaseModel {

protected $table = 'lan\_map';

private static $instance;

public static function getInstance(){

if (!self::$instance){

$firstMap = self::all()[0];

self::$instance = new Map($firstMap);

}

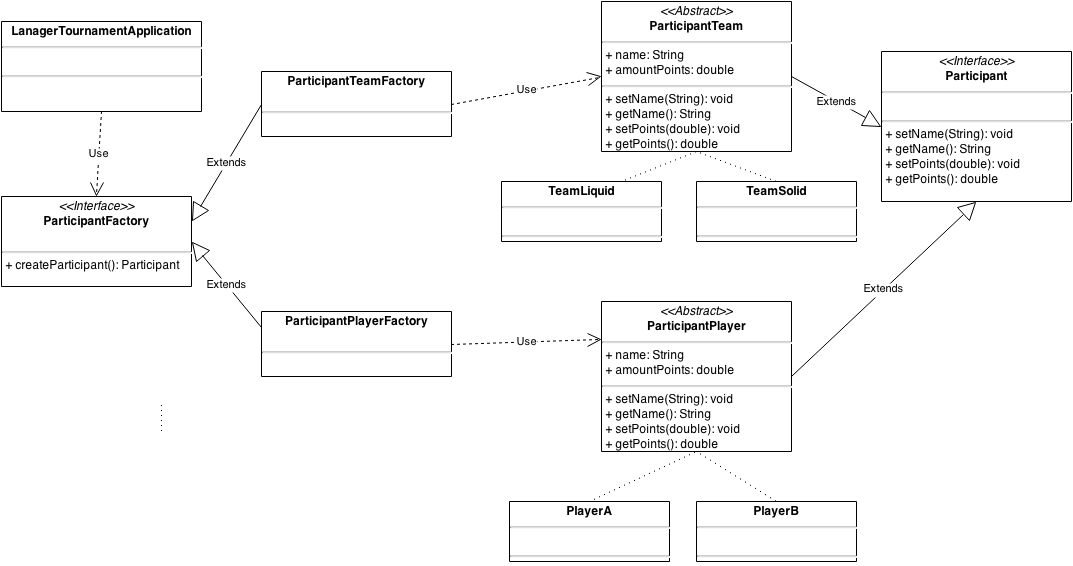
return self::$instance;

}

}

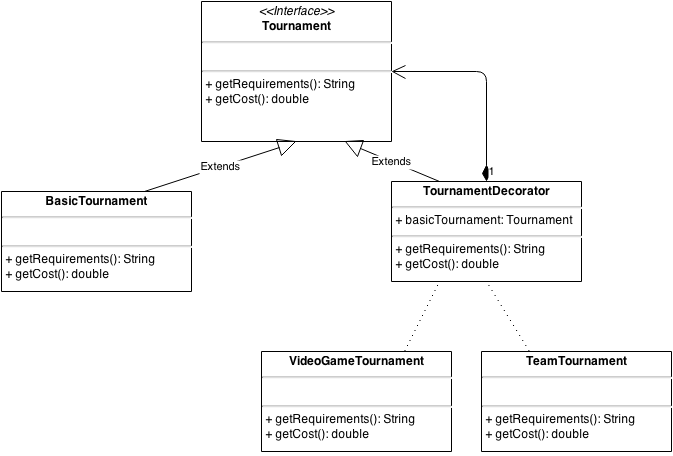
4.2. Absract Factory šablonas, skirtas dalyvių registracijai turnyruose. Turnyruose gali dalyvauti tiek žaidėjai, tiek komandos. Tam yra reikalinga atskirti šias esybes per abstract factory pattern’ą. Tiek komandos, tiek žaidėjai implementuoja Participant interface’ą.

Komandos ir žaidėjai yra abstraktūs - skirtingiems žaidimams reikalingos skirtingos sudeties komandos. tokios pačios sąlygos yra reikalingos ir žaidėjams. Skirtingiems žaidimams taikomi skirtingi žadiėjų tipai.



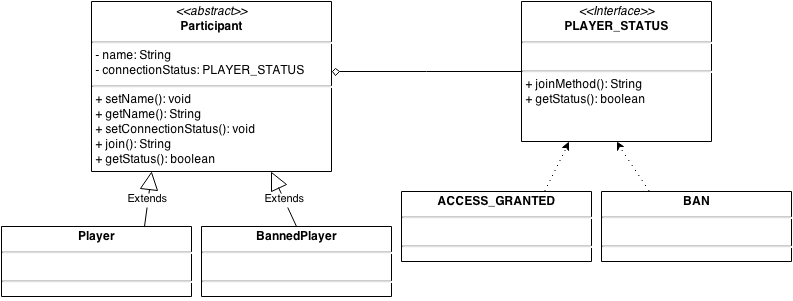
|  |
| --- |
| public class LanagerTeamsApplication{  public static void main(String[] args){  TournamentSingleton tournament = TournamentSingleton.getInstance();  ParticipantFactory factory1 = new ParticipantPlayerFactory();  tournament.addParticipant(factory1.createParticipant());  tournament.addParticipant(factory1.createParticipant());  tournament.addParticipant(factory1.createParticipant());  tournament.addParticipant(factory1.createParticipant());  System.out.println(tournament.getTournamentParticipants());  tournament.finishTournament();  ParticipantFactory factory2 = new ParticipantTeamFactory();  tournament.addParticipant(factory2.createParticipant());  tournament.addParticipant(factory2.createParticipant());  tournament.addParticipant(factory2.createParticipant());  tournament.addParticipant(factory2.createParticipant());  System.out.println(tournament.getTournamentParticipants());  tournament.finishTournament();  }  }  public interface Participant {  public void setName(String newName);  public String getName();  public void setPoints(double newDamage);  public double getPoints();  }  public interface ParticipantFactory {  Participant createParticipant();  }  public abstract class ParticipantPlayer implements Participant {    private String name;  private double amountDamage;    public void setName(String newName){  name = newName;  }  public String getName(){  return name;  }  public void setPoints(double newPoints){  amountDamage = newPoints;  }  public double getPoints() {  return amountDamage;  }  }  import java.io.BufferedReader;  import java.io.IOException;  import java.io.InputStreamReader;  import java.util.Scanner;  public class ParticipantPlayerFactory implements ParticipantFactory {  public Participant createPlayer(){  // Scanner sc = new Scanner(System.in);  System.out.println("Make a choise: A/B");  String userOption = "";  // if(sc.hasNextLine()){  // userOption = sc.nextLine();  // }  BufferedReader bufferRead = new BufferedReader(new InputStreamReader(System.in));  try {  userOption = bufferRead.readLine();  } catch (IOException e) {  e.printStackTrace();  }  Participant participantUnit = null;  if(userOption.equals("A")){  participantUnit = new PlayerA();  }else  if(userOption.equals("B")){  participantUnit = new PlayerB();  }  // sc.close();  return participantUnit;  }  @Override  public Participant createParticipant() {  // TODO Auto-generated method stub  return createPlayer();  }    }  public abstract class ParticipantTeam implements Participant {    private String name;  private double amountPoints;    public void setName(String newName){  name = newName;  }  public String getName(){  return name;  }  public void setPoints(double newPoints){  amountPoints = newPoints;  }  public double getPoints() {  return amountPoints;  }  }  import java.io.BufferedReader;  import java.io.IOException;  import java.io.InputStreamReader;  import java.util.Scanner;  public class ParticipantTeamFactory implements ParticipantFactory {  public Participant createTeam(){  //Scanner sc = new Scanner(System.in);  System.out.println("Make a choise: L/S");  String userOption = "";  // if(sc.hasNextLine()){  // userOption = sc.nextLine();  // }  BufferedReader bufferRead = new BufferedReader(new InputStreamReader(System.in));  try {  userOption = bufferRead.readLine();  } catch (IOException e) {  e.printStackTrace();  }    Participant participantUnit = null;  if(userOption.equals("L")){  participantUnit = new TeamLiquid();  }else  if(userOption.equals("S")){  participantUnit = new TeamSolid();  }  // sc.close();  return participantUnit;  }  @Override  public Participant createParticipant() {  // TODO Auto-generated method stub  return createTeam();  }    }  public class PlayerA extends ParticipantPlayer {    public PlayerA(){  setName("PlayerA");  setPoints(0);  }  }  public class PlayerB extends ParticipantPlayer {  public PlayerB(){  setName("PlayerB");  setPoints(0);  }    }  public class TeamLiquid extends ParticipantTeam {    public TeamLiquid(){  setName("Liquid Team");  setPoints(0);  }  }  public class TeamSolid extends ParticipantTeam {    public TeamSolid(){  setName("SolidTeam");  setPoints(0);  }  }  import java.util.ArrayList;  import java.util.List;  public class TournamentSingleton {  private static TournamentSingleton instance = null;    private List<Participant> participants;    private TournamentSingleton(){  this.participants = new ArrayList<Participant>();  }    public static TournamentSingleton getInstance(){  if(instance == null){  instance = new TournamentSingleton();  }  return instance;  }    public void addParticipant(Participant participant){  this.participants.add(participant);  }    public String getTournamentParticipants(){  String res = "";  for(Participant participant : this.participants){  res += participant.getName();  res += "\n";  }  return res;  }  public void finishTournament(){  this.participants.clear();  }    } |

4.3. Turnyrų Decorator. Renginyje gali būti įvairių tipų turnyrų. Stalo žaidimų turnyrai, Video žaidimų turnyrai, turnyrai komandoms… Kiekvieno tipo turnyras turi tiek bendrus reikalavimus, tačiau ir turi išskirtinių dalykų. Tam riekalingas dekoratorius - jog iš pagrindinės turnyro bazės sukurtume savo norimo tipo turnyrą.



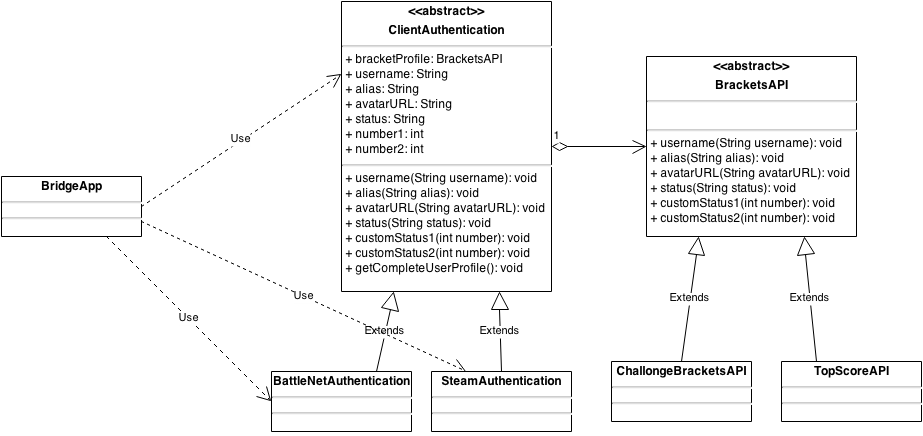
|  |
| --- |
| public class BasicTournament implements Tournament {  @Override  public String getRequirements() {  // TODO Auto-generated method stub  return "A computer";  }  @Override  public double getCost() {  // TODO Auto-generated method stub  return 10;  }  }  public class LanPartyEvents {  public static void main(String[] args) {  // TODO Auto-generated method stub    TournamentDecorator tournament = new TeamTournament(new VideoGameTournament(new BasicTournament()));    System.out.println(tournament.getRequirements());  System.out.println(tournament.getCost());    }  }  public interface Tournament {    public String getRequirements();    public double getCost();  }  public class TournamentDecorator implements Tournament {  Tournament basicTournament = null;    public TournamentDecorator(Tournament someTournament) {  basicTournament = someTournament;  }    @Override  public String getRequirements() {  // TODO Auto-generated method stub  return basicTournament.getRequirements();  }  @Override  public double getCost() {  // TODO Auto-generated method stub  return basicTournament.getCost();  }  }  public class TeamTournament extends TournamentDecorator {  public TeamTournament(Tournament someTournament) {  super(someTournament);  // TODO Auto-generated constructor stub  }    public String getRequirements(){  return super.getRequirements() + ", 5 team members";  }    public double getCost(){  return super.getCost() + 50;  }  }  public class VideoGameTournament extends TournamentDecorator {  public VideoGameTournament(Tournament someTournament) {  super(someTournament);  // TODO Auto-generated constructor stub  }    public String getRequirements(){  return super.getRequirements() + ", a video game";  }    public double getCost(){  return super.getCost() + 15;    }  } |

4.4. Serverio dalyvių Strategy. Prie žaidimo serverių gali prisijungti žaidėjai. Deja, ne visi gali būti priimti. Kai kurie dalyviai gali būti užblokuoti sistemos administratorių. Design pattern’o tikslas yra atskirti skirtingą dalyvių elgsena, kai jie yra užblokuoti arba atvirkščiai, kai jiems yra suteikta prieeiga prisijungti prie serverio.



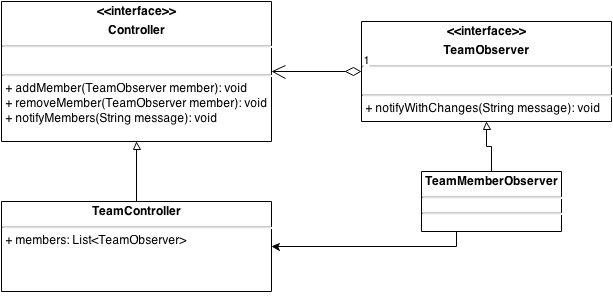
|  |
| --- |
| import java.util.ArrayList;  import java.util.List;  public class Server {  private static List<Participant> serverMembers;  public static void main(String[] args) {  serverMembers = new ArrayList<Participant>();  Participant player1 = new BannedPlayer();  player1.setName("Player1");  player1.setConnectionStatus(new BAN());  serverMembers.add(player1);  System.out.println(player1.join());  Participant player2 = new Player();  player2.setConnectionStatus(new ACCESS\_GRANTED());  player2.setName("Player2");  serverMembers.add(player2);  System.out.println(player2.join());  Participant player3 = new Player();  player3.setConnectionStatus(new ACCESS\_GRANTED());  player3.setName("Player3");  serverMembers.add(player3);  System.out.println(player3.join());  Participant player4 = new Player();  player4.setConnectionStatus(new ACCESS\_GRANTED());  player4.setName("Player4");  serverMembers.add(player4);  System.out.println(player4.join());  Participant player5 = new Player();  player5.setConnectionStatus(new ACCESS\_GRANTED());  player5.setName("Player5");  serverMembers.add(player5);  System.out.println(player5.join());  Participant player6 = new Player();  player6.setConnectionStatus(new ACCESS\_GRANTED());  player6.setName("Player6");  serverMembers.add(player6);  System.out.println(player6.join());  displayCurrentPlayers();  dismissBannedMembers();  displayCurrentPlayers();  }  private static void dismissBannedMembers(){  for(Participant participant: serverMembers){  if (!participant.getStatus()){  serverMembers.remove(participant);  }  }  }  private static void displayCurrentPlayers(){  for(Participant participant: serverMembers){  System.out.println(participant.getName());  }  }  }  public class Participant {    private String name;  private PLAYER\_STATUS connectionStatus;    public void setName(String newName){  name = newName;  }    public String getName(){  return name;  }    public void setConnectionStatus(PLAYER\_STATUS newConnectionStatus){  connectionStatus = newConnectionStatus;  }    public String join(){  return connectionStatus.joinMethod();  }    public boolean getStatus(){  return connectionStatus.getStatus();  }  }  public class Player extends Participant {    public Player(){  super();  setConnectionStatus(new ACCESS\_GRANTED());  }  }  public class BannedPlayer extends Participant {  public BannedPlayer(){  super();  setConnectionStatus(new BAN());  }  }  public interface PLAYER\_STATUS {    public String joinMethod();  public boolean getStatus();  }  public class ACCESS\_GRANTED implements PLAYER\_STATUS {  @Override  public String joinMethod() {  return "Joining a server....";  }  public boolean getStatus(){  return true;  }  }  public class BAN implements PLAYER\_STATUS {  @Override  public String joinMethod() {  return "Rejected. Reason: banned";  }  public boolean getStatus(){  return false;  }  } |

4.5. PĮ vartotojų autentifikacijos Strategy. Lanager PĮ palaiko vartotojų autentifikaciją per III-čių šalių platformas. Lanager palaiko daugelį autentifikacijos būdų, deja ne visi autentifikacijos būdai yra vienodi. Kai kurie būdai perduoda skirtingus informacijos kiekius. Visi autentifikacijos būdai perduoda vartotojo bazę - vardas, avataras, BIO ir t.t. Tačiau Steam prie vartotojo aprašymo perduoda papildomai ir Steam vartotojo inventorių. Battle.net pvz neturi tokios galimybės, tačiau gali perduoti informaciją, kurios Steam neturi.   
Sudarinėjant turnyrines lenteles, reikalinga turnyrinės lentelės sudarymo API. Šie servisai vėlgi, yra skirtingi. Challonge ir TopScore, priimantys bazinius duomenis, tačiau su kai kuriais iš jų besielgdami skirtingai.



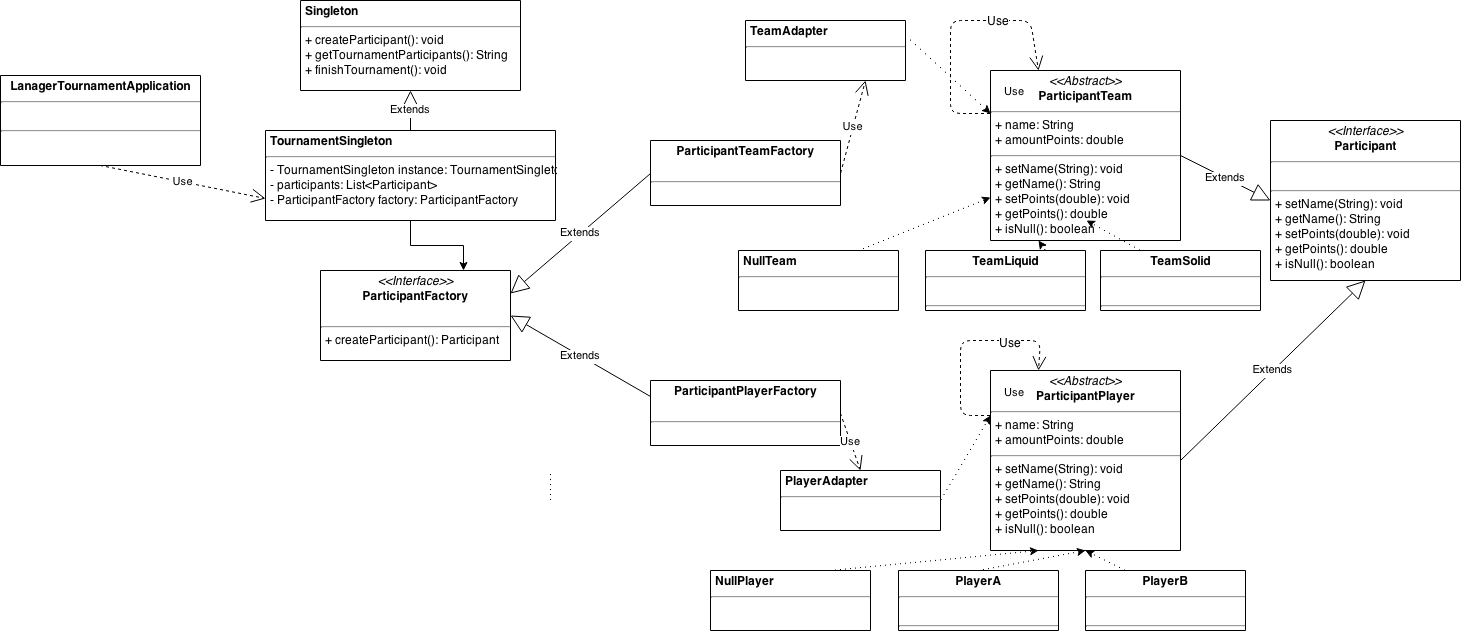
|  |
| --- |
| public class BridgeApp {  public static void main(String[] args){  ClientAuthentication player1 = new BattleNetAuthentication(new ChallongeBracketsAPI(), "Player1", "Captain", "url...", "active", 50, 500);  ClientAuthentication player2 = new SteamAuthentication(new TopScoreBracketsAPI(), "Player2", "Lead member", "url...", "offline", 25, 135);  ClientAuthentication player3 = new BattleNetAuthentication(new ChallongeBracketsAPI(), "Player3", "Member", "url...", "online", 50, 70);  ClientAuthentication player4 = new SteamAuthentication(new TopScoreBracketsAPI(), "Player4", "Recruit", "url...", "idle", 75, 25);  ClientAuthentication player5 = new BattleNetAuthentication(new ChallongeBracketsAPI(), "Player5", "Captain", "url...", "idle", 100, 30);  ClientAuthentication player6 = new SteamAuthentication(new TopScoreBracketsAPI(), "Player6", "Support", "url...", "active", 90, 50);  ClientAuthentication player7 = new SteamAuthentication(new TopScoreBracketsAPI(), "Player7", "Disabled", "url...", "active", 60, 30);  ClientAuthentication player8 = new BattleNetAuthentication(new ChallongeBracketsAPI(), "Player8", "Spare", "url...", "active", 40, 100);  ClientAuthentication player9 = new SteamAuthentication(new TopScoreBracketsAPI(), "Player9", "Member", "url...", "offline", 30, 135);  ClientAuthentication player10 = new BattleNetAuthentication(new ChallongeBracketsAPI(), "Player10", "Member", "url...", "active", 80, 50);  player1.getCompleteUserProfile();  System.out.println("---------------");  player2.getCompleteUserProfile();  System.out.println("---------------");  player3.getCompleteUserProfile();  System.out.println("---------------");  player4.getCompleteUserProfile();  System.out.println("---------------");  player5.getCompleteUserProfile();  System.out.println("---------------");  player6.getCompleteUserProfile();  System.out.println("---------------");  player7.getCompleteUserProfile();  System.out.println("---------------");  player8.getCompleteUserProfile();  System.out.println("---------------");  player9.getCompleteUserProfile();  System.out.println("---------------");  player10.getCompleteUserProfile();  System.out.println("---------------");  }    }  public class BattleNetAuthentication extends ClientAuthentication {  public BattleNetAuthentication(BracketsAPI bracketProfile, String username, String alias, String avatarUrl, String status, int number1, int number2) {  super(bracketProfile, username, alias, avatarUrl, status, number1, number2);  }  @Override  public void customStatus1(int number) {  System.out.println("Owned "+number+" games");  bracketProfile.customStatus1(number);  }  @Override  public void customStatus2(int number) {  System.out.println("Currently in "+number+" winstreak");  bracketProfile.customStatus2(number);  }  }  public abstract class BracketsAPI {    public BracketsAPI(){  }    public void username(String username){  System.out.println("Username is " + username);  }    public void alias(String alias){  System.out.println("Alias: " + alias);  }    public void avatarURL(String avatar){  System.out.println("Avatar URL: " + avatar);  }    public void status(String status){  System.out.println("User is currently " + status);  }    public abstract void customStatus1(int number);    public abstract void customStatus2(int number);  }  public class ChallongeBracketsAPI extends BracketsAPI {  public ChallongeBracketsAPI() {  super();  }  @Override  public void customStatus1(int number) {  System.out.println("<<C<< "+number+" >>C>>");  System.out.println("All player calculations and game information are trademarked under Challonge");  }  @Override  public void customStatus2(int number) {  System.out.println("<<C<< "+number+" >>C>>");  System.out.println("Trademark of Challonge");  }  }  public abstract class ClientAuthentication {  protected BracketsAPI bracketProfile = null;  protected String username;  protected String alias;  protected String avatarUrl;  protected String Status;  protected int number1;  protected int number2;  protected ClientAuthentication(BracketsAPI bracketProfile, String username, String alias, String avatarUrl, String status, int number1, int number2) {  this.bracketProfile = bracketProfile;  this.alias = alias;  this.username = username;  Status = status;  this.avatarUrl = avatarUrl;  this.number1 = number1;  this.number2 = number2;  }  public void username(String username){  bracketProfile.username(username);  }  public void alias(String alias){  bracketProfile.alias(alias);  }  public void avatarURL(String avatar){  bracketProfile.avatarURL(avatar);  }  public void status(String status){  bracketProfile.status(status);  }  abstract public void customStatus1(int number);  abstract public void customStatus2(int number);  public void getCompleteUserProfile(){  username(username);  alias(alias);  avatarURL(avatarUrl);  status(Status);  customStatus1(number1);  customStatus2(number2);  }  }  public class SteamAuthentication extends ClientAuthentication {  public SteamAuthentication(BracketsAPI bracketProfile, String username, String alias, String avatarUrl, String status, int number1, int number2) {  super(bracketProfile, username, alias, avatarUrl, status, number1, number2);  }  @Override  public void customStatus1(int number) {  System.out.println("Player since "+number+" year");  bracketProfile.customStatus1(number);  }  @Override  public void customStatus2(int number) {  System.out.println("Playing hours on record: "+number);  bracketProfile.customStatus2(number);  }  }  public class TopScoreBracketsAPI extends BracketsAPI {  public TopScoreBracketsAPI() {  super();  }  @Override  public void customStatus1(int number) {  System.out.println(number+" //TopScore TM");  System.out.println("No bets for this player yet");  }  @Override  public void customStatus2(int number) {  System.out.println(number+" //TopScore TM");  System.out.println("No bets for this player yet");  }  } |

4.6. Komandų įspėjimų Observer. Komandos darniam funkcionavimui reikalingi bet kokie pranešimai apie komandos sudėties pasikeitimą. Tam bus naudojamas Observer design pattern, jog kiekvienas komandos narys būtų informuotas, jeigu prie komandos prisijungė narys, arba iš jos buvo išmestas



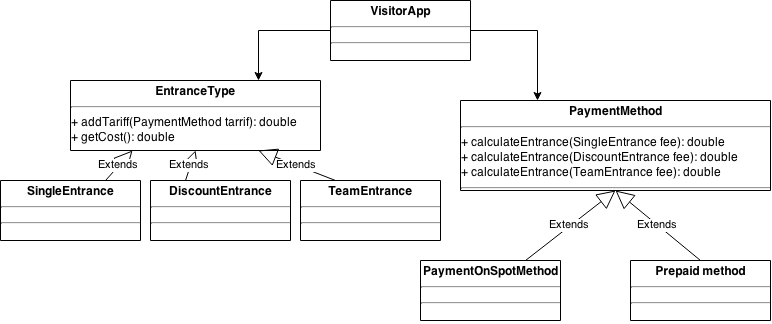
|  |
| --- |
| public class MainNotifications {    public static void main(String[] args){  TeamController team = new TeamController();  TeamObserver observer1 = new TeamMemberObserver();  TeamObserver observer2 = new TeamMemberObserver();  TeamObserver observer3 = new TeamMemberObserver();  TeamObserver observer4 = new TeamMemberObserver();  TeamObserver observer5 = new TeamMemberObserver();  TeamObserver observer6 = new TeamMemberObserver();  TeamObserver observer7 = new TeamMemberObserver();  TeamObserver observer8 = new TeamMemberObserver();  TeamObserver observer9 = new TeamMemberObserver();  TeamObserver observer10 = new TeamMemberObserver();  team.addMember(observer1);  team.addMember(observer2);  team.addMember(observer3);  team.addMember(observer4);  team.addMember(observer5);  team.removeMember(observer1);  team.removeMember(observer2);  team.removeMember(observer5);  team.addMember(observer6);  team.addMember(observer7);  team.addMember(observer8);  team.addMember(observer9);  team.addMember(observer10);  team.removeMember(observer9);  team.removeMember(observer4);  team.removeMember(observer10);  }  }  public interface Controller {    public void addMember(TeamObserver o);    public void removeMember(TeamObserver o);    public void notifyMembers(String message);  }  import java.util.ArrayList;  import java.util.List;  public class TeamController implements Controller {  private List<TeamObserver> members;  public TeamController() {  this.members = new ArrayList<TeamObserver>();  }  @Override  public void addMember(TeamObserver member) {  this.members.add(member);  notifyMembers("A new member has been added to the team");  }  @Override  public void removeMember(TeamObserver member) {  this.members.remove(member);  notifyMembers("A member has been disbanded from the team");  }  @Override  public void notifyMembers(String message) {  for(TeamObserver observer: this.members){  observer.notifyWithChanges(message);  }  }  }  public interface TeamObserver {    public void notifyWithChanges(String message);  }  // Represents each Observer that is monitoring changes in the subject  public class TeamMemberObserver implements TeamObserver {  @Override  public void notifyWithChanges(String message) {  System.out.println("Notification:" + message);  }  } |

4.7. Patobulintas turnyro Abstract Factory. Komandų ir žaidėjų įvedimas factory veikimo metu. Vartotojas gali įvesti vartotojo tipą, nežinomą factory šablonui. Šiuo atveju panaudotas Null Object design pattern. Jo pagalba grąžinamas null reikšmę turintis objektas. Taip pat padaryta, jog skaičiuojant turnyro sudėtį šiuos null objektus ignoruotų.  
Taip pat perkeltas Factory serviso naudojimas iš main programos naudojant dependancy injection. Nuo šiol turnyro Singleton kaip parametrą naudoja factory elementą, su kuriuo gali sukurti kokio tik nori tipo dalyvius pagrindinei programai nematant, koks factory yra naudojamas.

Renginys taip pat sutraukia užsienio komandas iš baltarusijos, lenkijos, rusijos. Skirtingos abecelės ir vartotojų komandų bei žaidėjų vardai sudaro problemas saugant intofrmaciją. Būtent todėl sukurtas Adapter šablonas, kuris valdo Player bei Team klases. Bet kokią informaciją išsaugant šiose klasėse, jos praeina pro adapter klasę, kuri konvertuoja reikšmes į UTF-8 formatą

|  |
| --- |
| public class LanagerTeamsApplication{  public static void main(String[] args){  TournamentSingleton tournament = TournamentSingleton.getInstance(new ParticipantPlayerFactory());  tournament.createParticipant();  tournament.createParticipant();  tournament.createParticipant();  tournament.createParticipant();  tournament.createParticipant();  tournament.createParticipant();  System.out.println(tournament.getTournamentParticipants());  tournament.finishTournament();  tournament = TournamentSingleton.getInstance(new ParticipantTeamFactory());  tournament.createParticipant();  tournament.createParticipant();  tournament.createParticipant();  tournament.createParticipant();  tournament.createParticipant();  tournament.createParticipant();  System.out.println(tournament.getTournamentParticipants());  tournament.finishTournament();  }  }  /\*\*  \* Created by petkus on 12/11/14.  \*/  public class NullPlayer extends ParticipantPlayer {  public NullPlayer(){  setName("No Player type found");  }  @Override  public boolean isNull(){  return true;  }  }  public class NullTeam extends ParticipantTeam {  public NullTeam(){  setName("No Team type found");  }  @Override  public boolean isNull(){  return true;  }  }  public interface Participant {  public void setName(String newName);  public String getName();  public void setPoints(double newDamage);  public double getPoints();  public boolean isNull();  }  public interface ParticipantFactory {  Participant createParticipant();  }  public abstract class ParticipantPlayer implements Participant {    private String name;  private double amountDamage;    public void setName(String newName){  name = newName;  }  public String getName(){  return name;  }  public void setPoints(double newPoints){  amountDamage = newPoints;  }  public double getPoints() {  return amountDamage;  }  @Override  public boolean isNull(){  return true;  }  }  import java.io.BufferedReader;  import java.io.IOException;  import java.io.InputStreamReader;  import java.util.Scanner;  public class ParticipantPlayerFactory implements ParticipantFactory {  public Participant createPlayer(){  // Scanner sc = new Scanner(System.in);  System.out.println("Make a choise: A/B");  String userOption = "";  // if(sc.hasNextLine()){  // userOption = sc.nextLine();  // }  BufferedReader bufferRead = new BufferedReader(new InputStreamReader(System.in));  try {  userOption = bufferRead.readLine();  } catch (IOException e) {  e.printStackTrace();  }  Participant participantUnit = null;  if(userOption.equals("A")){  participantUnit = new PlayerAdapter(new PlayerA());  }else  if(userOption.equals("B")){  participantUnit = new PlayerAdapter(new PlayerB());  }  else{  participantUnit = new PlayerAdapter(new NullPlayer());  }  // sc.close();  return participantUnit;  }  @Override  public Participant createParticipant() {  // TODO Auto-generated method stub  return createPlayer();  }    }  public abstract class ParticipantTeam implements Participant {    private String name;  private double amountPoints;  public void setName(String newName){  name = newName;  }  public String getName(){  return name;  }  public void setPoints(double newPoints){  amountPoints = newPoints;  }  public double getPoints() {  return amountPoints;  }  @Override  public boolean isNull(){  return true;  }  }  import java.io.BufferedReader;  import java.io.IOException;  import java.io.InputStreamReader;  import java.util.Scanner;  public class ParticipantTeamFactory implements ParticipantFactory {  public Participant createTeam(){  //Scanner sc = new Scanner(System.in);  System.out.println("Make a choise: L/S");  String userOption = "";  // if(sc.hasNextLine()){  // userOption = sc.nextLine();  // }  BufferedReader bufferRead = new BufferedReader(new InputStreamReader(System.in));  try {  userOption = bufferRead.readLine();  } catch (IOException e) {  e.printStackTrace();  }    Participant participantUnit = null;  if(userOption.equals("L")){  participantUnit = new TeamAdapter(new TeamLiquid());  }else  if(userOption.equals("S")){  participantUnit = new TeamAdapter(new TeamSolid());  }  else{  participantUnit = new TeamAdapter(new NullTeam());  }  // sc.close();  return participantUnit;  }  @Override  public Participant createParticipant() {  // TODO Auto-generated method stub  return createTeam();  }    }  public class PlayerA extends ParticipantPlayer {    public PlayerA(){  setName("PlayerA");  setPoints(0);  }  @Override  public boolean isNull(){  return false;  }  }  import java.io.UnsupportedEncodingException;  /\*\*  \* Created by petkus on 12/12/14.  \*/  public class PlayerAdapter extends ParticipantPlayer{  ParticipantPlayer player;  public PlayerAdapter(ParticipantPlayer player) {  this.player = player;  }  public void setName(String newName){  byte ptext[] = newName.getBytes();  String value = null;  try {  value = new String(ptext, "UTF-8");  } catch (UnsupportedEncodingException e) {  e.printStackTrace();  }  this.player.setName(value);  }  public String getName(){  return player.getName();  }  public void setPoints(double newPoints){  if (newPoints > 0 && newPoints < 100){  player.setPoints(newPoints);  }  else{  player.setPoints(0);  }  }  public double getPoints() {  return player.getPoints();  }  @Override  public boolean isNull(){  return player.isNull();  }  }  public class PlayerB extends ParticipantPlayer {   public PlayerB(){  setName("PlayerB");  setPoints(0);  }   @Override  public boolean isNull(){  return false;  } } /\*\*  \* Created by petkus on 12/11/14.  \*/  public interface Singleton {  public void createParticipant();  public String getTournamentParticipants();  public void finishTournament();  }  public class TeamLiquid extends ParticipantTeam {    public TeamLiquid(){  setName("Liquid Team");  setPoints(0);  }  @Override  public boolean isNull(){  return false;  }  }  public class TeamSolid extends ParticipantTeam {    public TeamSolid(){  setName("SolidTeam");  setPoints(0);  }  @Override  public boolean isNull(){  return false;  }  }  import java.util.ArrayList;  import java.util.List;  public class TournamentSingleton implements Singleton {  private static TournamentSingleton instance = null;    private List<Participant> participants;  private ParticipantFactory factory;    private TournamentSingleton(ParticipantFactory fac){  this.participants = new ArrayList<Participant>();  this.factory = fac;  }    public static TournamentSingleton getInstance(ParticipantFactory fac){  if(instance == null){  instance = new TournamentSingleton(fac);  }  return instance;  }    public void createParticipant(){  this.participants.add(this.factory.createParticipant());  }    public String getTournamentParticipants(){  String res = "";  for(Participant participant : this.participants){  if (!participant.isNull()){  res += participant.getName();  res += "\n";  }  }  return res;  }  public void finishTournament(){  this.participants.clear();  }    }  import java.io.UnsupportedEncodingException;  /\*\*  \* Created by petkus on 12/12/14.  \*/  public class PlayerAdapter extends ParticipantPlayer{  ParticipantPlayer player;  public PlayerAdapter(ParticipantPlayer player) {  this.player = player;  }  public void setName(String newName){  byte ptext[] = newName.getBytes();  String value = null;  try {  value = new String(ptext, "UTF-8");  } catch (UnsupportedEncodingException e) {  e.printStackTrace();  }  this.player.setName(value);  }  public String getName(){  return player.getName();  }  public void setPoints(double newPoints){  if (newPoints > 0 && newPoints < 100){  player.setPoints(newPoints);  }  else{  player.setPoints(0);  }  }  public double getPoints() {  return player.getPoints();  }  @Override  public boolean isNull(){  return player.isNull();  }  }  import java.io.UnsupportedEncodingException;  /\*\*  \* Created by petkus on 12/12/14.  \*/  public class TeamAdapter extends ParticipantTeam{  ParticipantTeam team;  public TeamAdapter(ParticipantTeam team) {  this.team = team;  }  public void setName(String newName){  byte ptext[] = newName.getBytes();  String value = null;  try {  value = new String(ptext, "UTF-8");  } catch (UnsupportedEncodingException e) {  e.printStackTrace();  }  this.team.setName(value);  }  public String getName(){  return team.getName();  }  public void setPoints(double newPoints){  if (newPoints > 0 && newPoints < 100){  team.setPoints(newPoints);  }  else{  team.setPoints(0);  }  }  public double getPoints() {  return team.getPoints();  }  @Override  public boolean isNull(){  return team.isNull();  }  } |

4.8. Apmokėjimo visitor. Yra dvejų rūšių apmokėjimai - išankstinis, kai bilietai yra pigesni. Ir yra apmokėjimas vietoje, kai bilietas yra brangesnis. Taip pat yra kelios bilietų rūšys priklausančios nuo kitų nei apmokėjimo laiko faktorių - ar žmogus atkeliavo vienas, ar turi specialų nuolaidos kuponą, ar atėjo su komanda, kuriai taikoma nuolaida bilietui. Į šias nuolaidas, lengvatas taip pat turima atsižvelgi visais atžvilgiais ir kai apmokama iš anksto, ir kai apmokama vietoje. Tam reikia Visitor šablono



|  |
| --- |
| public class VisitorApp {  public static void main(String[] args){    SingleEntrance method1 = new SingleEntrance(25);  DiscountEntrance method2 = new DiscountEntrance(20);  TeamEntrance method3 = new TeamEntrance(22);    PrepaidMethod prepaidMethod = new PrepaidMethod();    System.out.println("Prepaid entrance for single person: " + method1.addTariff(prepaidMethod) );  System.out.println("Prepaid entrance with discount: " + method2.addTariff(prepaidMethod) );  System.out.println("Prepaid entrance with team: " + method3.addTariff(prepaidMethod) );    PaymentOnSpotMethod paymentOnSpotMethod = new PaymentOnSpotMethod();  System.out.println("Paid on spot entrance for single person: " + method1.addTariff(paymentOnSpotMethod) );  System.out.println("Paid on spot entrance with discount: " + method2.addTariff(paymentOnSpotMethod) );  System.out.println("Paid on spot entrance with team: " + method3.addTariff(paymentOnSpotMethod) );      }    }  public interface EntranceType {  public double addTariff(PaymentMethod tariff);    }  public class DiscountEntrance implements EntranceType {  double baseCost = 0;    public DiscountEntrance(double price){  baseCost = price;  System.out.println("Entrance with discount cost: " + baseCost);    }    public double getCost(){  return baseCost;  }  @Override  public double addTariff(PaymentMethod tariff) {  return tariff.calculateEntrance(this);  }    }  public class TeamEntrance implements EntranceType {  double baseCost = 0;    public TeamEntrance(double price){  baseCost = price;  System.out.println("Team entrance cost: " + baseCost);  }    public double etCost(){  return baseCost;  }  @Override  public double addTariff(PaymentMethod tariff) {  return tariff.calculateEntrance(this);  }      }  public class SingleEntrance implements EntranceType {   double baseCost = 0;    public SingleEntrance(double price){  baseCost = price;  System.out.println("Single person cost: " + baseCost);  }    public double getCost(){  return baseCost;  }   @Override  public double addTariff(PaymentMethod tariff) {  return tariff.calculateEntrance(this);  }      }  public interface PaymentMethod {  public double calculateEntrance(SingleEntrance fee);  public double calculateEntrance(DiscountEntrance fee);  public double calculateEntrance(TeamEntrance fee);    }  public class PaymentOnSpotMethod implements PaymentMethod {  @Override  public double calculateEntrance(SingleEntrance fee) {  return fee.getCost() + 10;    }  @Override  public double calculateEntrance(DiscountEntrance fee) {  return fee.getCost() + 0;  }  @Override  public double calculateEntrance(TeamEntrance fee) {  return fee.etCost() + 5;  }  }  public class PrepaidMethod implements PaymentMethod {  @Override  public double calculateEntrance(SingleEntrance fee) {  return fee.getCost() + 5;    }  @Override  public double calculateEntrance(DiscountEntrance fee) {  return fee.getCost() - 5;  }  @Override  public double calculateEntrance(TeamEntrance fee) {  return fee.etCost() + 0;  }  } |