|  |
| --- |
| E:\KTU\ktu-ikona.png  **KAUNO TECHNOLOGIJOS UNIVERSITETAS**  **INFORMATIKOS FAKULTETAS**    **Objektinis programų projektavimas (T120B516)**  **Projekto užduoties Nr. 3 ataskaita**  **Darbą atliko IFF-7/14 Grupės studentai:**  Rimvydas Neverauskas  Valentinas Kasteckis  Henrikas Juzuitis  Eligijus Kiudys  **Darbą priėmė:**  prof. BAREIŠA Eduardas  lekt. UŠANIOV Andrej  Kaunas, 2020 |

Turinys

[Paveikslėlių sąrašas 3](#_Toc58616861)

[1. Įvadas 4](#_Toc58616862)

[1.1. Darbo tikslas 4](#_Toc58616863)

[1.2. Projekto aprašymas 4](#_Toc58616864)

[1.3. Panaudojimo atvejų diagrama 4](#_Toc58616865)

[2. Pirmojo etapo projektavimo šablonai 4](#_Toc58616866)

[2.1. Singleton 4](#_Toc58616867)

[2.2. Factory 5](#_Toc58616868)

[2.3. AbstractFactory 8](#_Toc58616869)

[2.4. Strategy 13](#_Toc58616870)

[2.5. Observer 16](#_Toc58616871)

[2.6. Builder 24](#_Toc58616872)

[2.7. Prototype 28](#_Toc58616873)

[2.8. Decorator 29](#_Toc58616874)

[2.9. Command 33](#_Toc58616875)

[2.10. Adapter 37](#_Toc58616876)

[2.11. Facade 38](#_Toc58616877)

[2.12. Bridge 51](#_Toc58616878)

[3. Antrojo etapo projektavimo šablonai 54](#_Toc58616879)

[3.1. Template method 54](#_Toc58616880)

[3.2. Iterator 57](#_Toc58616881)

[3.3. Flyweight 61](#_Toc58616882)

[3.4. Composite 62](#_Toc58616883)

[3.5. State 63](#_Toc58616884)

[3.6. Proxy 64](#_Toc58616885)

[3.7. Chain Of Responsibility 64](#_Toc58616886)

[3.8. Visitor 65](#_Toc58616887)

[3.9. Interpreter 65](#_Toc58616888)

[3.10. Mediator 66](#_Toc58616889)

[3.11. Memento 66](#_Toc58616890)

[4. Išvados 68](#_Toc58616891)

# Paveikslėlių sąrašas

[Pav. 1 panaudojimo atvejų diagrama 4](#_Toc58616892)

[Pav. 2 singleton šablono diagrama 4](#_Toc58616893)

[Pav. 3 factory method šablono diagramos fragmentas 6](#_Toc58616894)

[Pav. 4 abstract factory šablono diagramos fragmentas 8](#_Toc58616895)

[Pav. 5 strategy šablono diagramos fragmentas 14](#_Toc58616896)

[Pav. 6 observer šablono diagramos fragmentas 16](#_Toc58616897)

[pav. 7 observer šablono sekų diagrama 17](#_Toc58616898)

[Pav. 8 builder šablono kodo fragmentas 24](#_Toc58616899)

[Pav. 9 prototype šablono diagramos fragmentas 28](#_Toc58616900)

[Pav. 10 prototype šablono pavyzdys 29](#_Toc58616901)

[Pav. 11 objektų numeriai 29](#_Toc58616902)

[Pav. 12 decorator šablono diagramos fragmentas 30](#_Toc58616903)

[Pav. 13 command šablono diagramos fragmentas 34](#_Toc58616904)

[Pav. 14 adapter šablono diagramos fragmentas 37](#_Toc58616905)

[Pav. 15 facade šablono diagramos fragmentas 39](#_Toc58616906)

[Pav. 16 bridge šablono diagramos fragmentas 51](#_Toc58616907)

[pav. 17 template method šablono diagramos fragmentas 54](#_Toc58616908)

[pav. 18 iterator šablono diagramos fragmentas 57](#_Toc58616909)

[pav. 19 flyweight šablono diagramos fragmentas 61](#_Toc58616910)

[pav. 20 flyweight šablono greitaveikos ir atminties naudojimo matavimas 61](#_Toc58616911)

[pav. 21 composite 6ablono diagramos fragmentas 62](#_Toc58616912)

[pav. 22 state šablono diagramos fragmentas 63](#_Toc58616913)

[pav. 23 proxy šablono diagramos fragmentas 64](#_Toc58616914)

[pav. 24 chain of responsibility šablono diagramos fragmentas 64](#_Toc58616915)

[pav. 25 visitor šablono šablono diagramos fragmentas 65](#_Toc58616916)

[pav. 26 interpreter šablono diagramos fragmentas 65](#_Toc58616917)

[pav. 27 mediator šablono diagramos fragmentas 66](#_Toc58616918)

[pav. 28 memento šablono diagramos fragmentas 66](#_Toc58616919)

# Įvadas

## Darbo tikslas

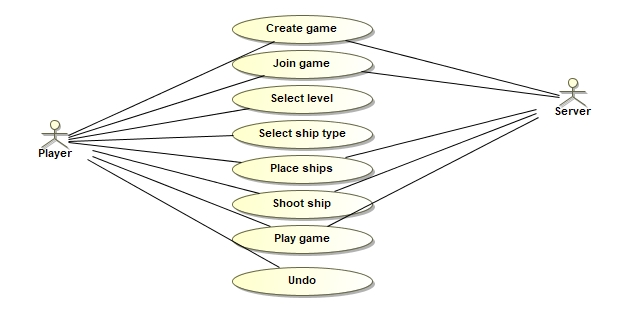
Darbo tikslas – kuriant 2D daugelio žaidėjų tinklo žaidimą (multiplayer network game), išmokti taikyti projektavimo šablonus (Design Patterns) bei susipažinti su jų naudojimo ypatumais. Darbas turi būti tęstinis, papildomas, plečiamas bei tobulinamas semestro eigoje.

## Projekto aprašymas

Kuriamas 2D žaidimas – laivų mūšis. Šio žaidimo kūrimui naudojame **Java** programavimo kalbą. Komunikavimui tarp žaidėjų yra naudojami **REST** web service principai, bei duomenys yra perduodami **JSON** formatu. Tam tikri duomenys yra saugomi duomenų bazėje (pvz.: žaidėjo ir žaidimo ID). Žaidimo vaizdavimui yra naudojama Java biblioteka „**swing**“.

## Panaudojimo atvejų diagrama

Pav. 1 panaudojimo atvejų diagrama

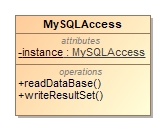


# Pirmojo etapo projektavimo šablonai

## Singleton

Pritaikę šį šabloną duomenų bazės prieeigai, užtikriname kad bus tik vienas priėjimas.

Pav. 2 singleton šablono diagrama



Šis šablonas realizuotas su „thread safe“. Ši klasė yra atsakinga yra už komunikacija su MySQL, ji yra viena vienintelė per visą runtime periodą, visos klasės naudojasi tik vienu objektu.

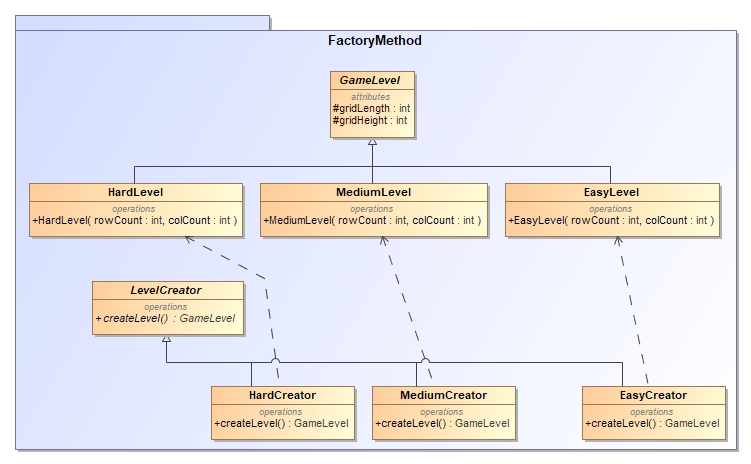
Šio šablono kodas:

**public class** MysqlAccess {  
 **public static** String *SERVER\_IP* = Dotenv.*load*().get(**"SERVER\_IP"**) + **":"** + Dotenv.*load*().get(**"DATABASE\_PORT"**);  
 **public static** String *USERNAME* = Dotenv.*load*().get(**"DATABASE\_USERNAME"**);  
 **public static** String *PASSWORD* = Dotenv.*load*().get(**"DATABASE\_PASSWORD"**);  
  
 **public** Connection **connection** = **null**;  
 **private static** MysqlAccess *instance* = **new** MysqlAccess();  
  
 **public static** MysqlAccess getInstance()  
 {  
 **return** MysqlAccess.*instance*;  
 }  
  
 **public void** readDataBase() **throws** Exception {  
 String url = **"jdbc:mysql://"** + *SERVER\_IP* + **"/project"**; *// live portas 3306 local: 9906* System.***out***.println(**"Connecting database..."**);  
  
 **try** {  
 **connection** = DriverManager.*getConnection*(url, *USERNAME*, *PASSWORD*);  
 System.***out***.println(**"Database connected!"**);  
 } **catch** (SQLException e) {  
 **throw new** IllegalStateException(**"Cannot connect the database!"**, e);  
 }  
 }  
  
 **public** ArrayList<String> writeResultSet(ResultSet resultSet) **throws** SQLException {  
 ArrayList<String> arrayList = **new** ArrayList<>();  
 **while** (resultSet.next()) {  
 arrayList.add(resultSet.getString(**"test"**));  
 }  
  
 **return** arrayList;  
 }  
}

## Factory

Kadangi turime 3 skirtingus žaidimo lygius (Easy, Medium, Hard) pritaikėme šį šabloną žaidimo lygių kūrimui.

Pav. 3 factory method šablono diagramos fragmentas



FactoryMethod žaidime buvo panaudotas, kaip skirtingų žaidimo lygių kūrėjas. Žaidėjas prieš sukurdamas žaidimo sesija, turi pasirinkti norimą žaidimo lygį. Nuo žaidimo lygio priklauso žaidimo jūros matricos (Grid) dydis pvz. (10x10).

**public abstract class** GameLevel  
{  
 **protected int gridLength**;  
   
 **protected int gridHeight**;  
  
 **public int** getGridLength() {  
 **return gridLength**;  
 }  
  
 **public int** getGridHeight() {  
 **return gridHeight**;  
 }  
}

**public class** EasyLevel **extends** GameLevel  
{  
 **public** EasyLevel(**int** rowCount, **int** colCount) {  
 **super**();  
  
 **this**.**gridHeight** = rowCount;  
 **this**.**gridLength** = colCount;  
 }  
}

**public class** MediumLevel **extends** GameLevel  
{  
 **public** MediumLevel(**int** rowCount, **int** colCount) {  
 **super**();  
  
 **this**.**gridHeight** = rowCount;  
 **this**.**gridLength** = colCount;  
 }  
}

**public class** HardLevel **extends** GameLevel  
{  
 **public** HardLevel(**int** rowCount, **int** colCount) {  
 **super**();  
  
 **this**.**gridHeight** = rowCount;  
 **this**.**gridLength** = colCount;  
 }  
}

**public abstract class** LevelCreator  
{  
 **public abstract** GameLevel createLevel();  
}

**public class** EasyCreator **extends** LevelCreator  
{  
 **public** GameLevel createLevel( )  
 {  
 **return new** EasyLevel(ComponentsSizeConstants.***easyRowCount***, ComponentsSizeConstants.***easyColCount***);  
 }  
}

**public class** MediumCreator **extends** LevelCreator  
{  
 **public** GameLevel createLevel( )  
 {  
 **return new** MediumLevel(ComponentsSizeConstants.***mediumRowCount***, ComponentsSizeConstants.***mediumColCount***);  
 }  
}

**public class** HardCreator **extends** LevelCreator  
{  
 **public** GameLevel createLevel( )  
 {  
 **return new** HardLevel(ComponentsSizeConstants.***hardRowCount***, ComponentsSizeConstants.***hardColCount***);  
 }  
}

**public class** GameController **implements** SeaGridPanelDelegate, MenuPanelDelegate, ChatPanelDelegate {

//\*\*CODE\*\*

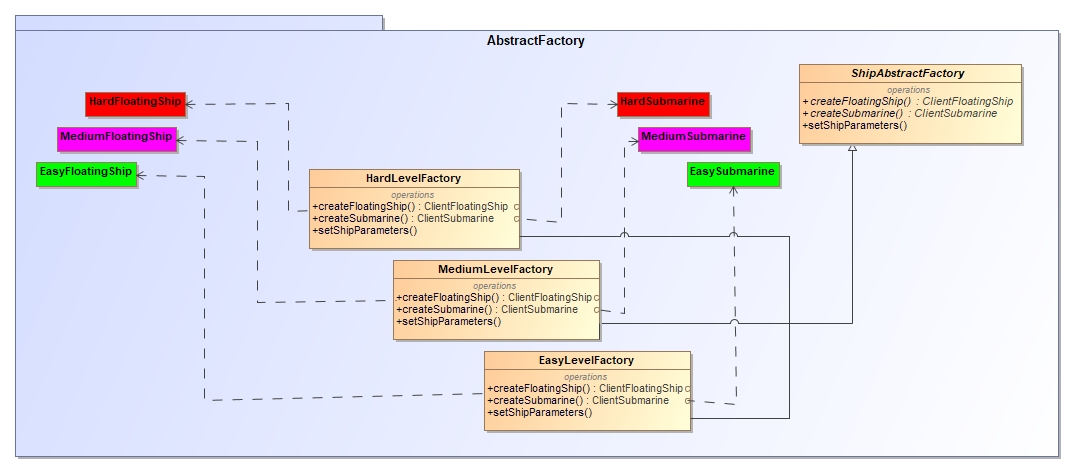
**private void** setGameLevel(String gameLevel){  
 **switch** (gameLevel){  
 **case** (GameConstants.***gameLevelEasy***):  
 GameData.*shipAbstractFactory* = **new** EasyLevelFactory();  
 GameData.*levelFactory* = **new** EasyCreator();  
 GameData.*SelectedGameLevel* = GameConstants.***gameLevelEasy***;  
 **break**;  
 **case** (GameConstants.***gameLevelMedium***):  
 GameData.*shipAbstractFactory* = **new** MediumLevelFactory();  
 GameData.*levelFactory* = **new** MediumCreator();  
 GameData.*SelectedGameLevel* = GameConstants.***gameLevelMedium***;  
 **break**;  
 **case** (GameConstants.***gameLevelHard***):  
 GameData.*shipAbstractFactory* = **new** HardLevelFactory();  
 GameData.*levelFactory* = **new** HardCreator();  
 GameData.*SelectedGameLevel* = GameConstants.***gameLevelHard***;  
 **break**;  
 }  
}

}

## AbstractFactory

Panaudodami šį šabloną praplėtėme „Factory“ šabloną.

Pav. 4 abstract factory šablono diagramos fragmentas



AbstractFactory žaidime buvo panaudotas, kaip skirtingų žaidimo lygių laivų kūrėjas. Žaidime skirtinguose lygiuose yra skirtingi laivai pvz. (Povandeninio laivo ilgis gali skirtis tarp žaidimo lygių).

**public abstract class** ShipAbstractFactory  
{  
 **protected** FloatingShipBuilder **floatingShipBuilder**;  
 **protected** SubmarineBuilder **submarineBuilder**;  
  
 **protected** Dictionary **shipsMade**;  
 **protected** Dictionary **shipParameters**;  
  
 **public abstract** ClientFloatingShip createFloatingShip(ShipType type);  
   
 **public abstract** ClientSubmarine createSubmarine();  
   
 **public** ShipAbstractFactory(){  
 **floatingShipBuilder** = **new** FloatingShipBuilder();  
 **submarineBuilder** = **new** SubmarineBuilder();  
 **shipsMade** = **new** Hashtable();  
 **shipParameters** = **new** Hashtable();  
 }  
  
 **public abstract** ShipParameters getShipParameters(ShipType type);  
  
 **public void** setShipRotation(ClientShip ship)  
 {  
 ship.setRotation(GameData.*Rotation*);  
 }  
}

**public class** HardLevelFactory **extends** ShipAbstractFactory  
{  
  
 **public** HardLevelFactory(){  
 **super**();  
 setShipParameters();  
 }  
  
 **private void** setShipParameters(){  
 **for** (ShipType type: GameConstants.***shipTypes***) {  
 **this**.**shipParameters**.put(type, GameConstants.HardLevelShips.*getShipParameters*(type));  
 }  
 }  
  
 **public** ClientFloatingShip createFloatingShip(ShipType type)  
 {  
 HardFloatingShip madeShip = (HardFloatingShip) **shipsMade**.get(type);  
 **if** (madeShip != **null**){  
 HardFloatingShip clone = (HardFloatingShip) madeShip.makeCopy();  
 **return** clone;  
 }**else** {  
 ShipParameters shipParameters = **this**.getShipParameters(type);  
  
 HardFloatingShip hardFloatingShip = **new** HardFloatingShip(type);  
 setShipRotation(hardFloatingShip);  
  
 ShipConfigurator configurator = **floatingShipBuilder**.startHard(hardFloatingShip);  
 **for** (**int** i = 0; i < shipParameters.**shipLength**; i++) {  
 configurator.addShipPart();  
 }  
  
 **if** (shipParameters.**shield**){  
 configurator.addShipShield();  
 }  
  
 madeShip = (HardFloatingShip) configurator.build();  
 **shipsMade**.put(type, madeShip.makeCopy());  
  
 **return** madeShip;  
 }  
 }  
   
 **public** ClientSubmarine createSubmarine()  
 {  
 HardSubmarine madeShip = (HardSubmarine) **shipsMade**.get(ShipType.***SUBMARINE***);  
 **if** (madeShip != **null**){  
 HardSubmarine clone = (HardSubmarine) madeShip.makeCopy();  
 **return** clone;  
 }**else** {  
 ShipParameters shipParameters = **this**.getShipParameters(ShipType.***SUBMARINE***);  
  
 HardSubmarine hardSubmarine = **new** HardSubmarine();  
 setShipRotation(hardSubmarine);  
  
 ShipConfigurator configurator = **submarineBuilder**.startHard(hardSubmarine);  
 **for** (**int** i = 0; i < shipParameters.**shipLength**; i++) {  
 configurator.addShipPart();  
 }  
  
 **if** (shipParameters.**shield**){  
 configurator.addShipShield();  
 }  
  
 madeShip = (HardSubmarine) configurator.build();  
 **shipsMade**.put(ShipType.***SUBMARINE***, madeShip.makeCopy());  
  
 **return** madeShip;  
 }  
 }  
  
 @Override  
 **public** ShipParameters getShipParameters(ShipType type) {  
 **return** (ShipParameters) **this**.**shipParameters**.get(type);  
 }  
}

**public class** MediumLevelFactory **extends** ShipAbstractFactory  
{  
  
 **public** MediumLevelFactory(){  
 **super**();  
 setShipParameters();  
 }  
  
 **private void** setShipParameters(){  
 **for** (ShipType type: GameConstants.***shipTypes***) {  
 **this**.**shipParameters**.put(type, GameConstants.MediumLevelShips.*getShipParameters*(type));  
 }  
 }  
  
 **public** ClientFloatingShip createFloatingShip(ShipType type)  
 {  
 MediumFloatingShip madeShip = (MediumFloatingShip) **shipsMade**.get(type);  
 **if** (madeShip != **null**){  
 MediumFloatingShip clone = (MediumFloatingShip) madeShip.makeCopy();  
 **return** clone;  
 }**else** {  
 ShipParameters shipParameters = **this**.getShipParameters(type);  
 MediumFloatingShip mediumFloatingShip = **new** MediumFloatingShip(type);  
 setShipRotation(mediumFloatingShip);  
  
 ShipConfigurator configurator = **floatingShipBuilder**.startMedium(mediumFloatingShip);  
 **for** (**int** i = 0; i < shipParameters.**shipLength**; i++) {  
 configurator.addShipPart();  
 }  
  
 **if** (shipParameters.**shield**){  
 configurator.addShipShield();  
 }  
  
 madeShip = (MediumFloatingShip) configurator.build();  
 **shipsMade**.put(type, madeShip.makeCopy());  
  
 **return** madeShip;  
 }  
 }  
   
 **public** ClientSubmarine createSubmarine()  
 {  
 MediumSubmarine madeShip = (MediumSubmarine) **shipsMade**.get(ShipType.***SUBMARINE***);  
 **if** (madeShip != **null**){  
 MediumSubmarine clone = (MediumSubmarine) madeShip.makeCopy();  
 **return** clone;  
 }**else** {  
 ShipParameters shipParameters = **this**.getShipParameters(ShipType.***SUBMARINE***);  
 MediumSubmarine mediumSubmarine = **new** MediumSubmarine();  
 setShipRotation(mediumSubmarine);  
  
 ShipConfigurator configurator = **submarineBuilder**.startMedium(mediumSubmarine);  
 **for** (**int** i = 0; i < shipParameters.**shipLength**; i++) {  
 configurator.addShipPart();  
 }  
  
 **if** (shipParameters.**shield**){  
 configurator.addShipShield();  
 }  
  
 madeShip = (MediumSubmarine) configurator.build();  
 **shipsMade**.put(ShipType.***SUBMARINE***, madeShip.makeCopy());  
  
 **return** madeShip;  
 }  
 }  
   
 @Override  
 **public** ShipParameters getShipParameters(ShipType type) {  
 **return** (ShipParameters) **this**.**shipParameters**.get(type);  
 }  
}

**public class** EasyLevelFactory **extends** ShipAbstractFactory  
{  
  
 **public** EasyLevelFactory(){  
 **super**();  
 setShipParameters();  
 }  
  
 **public** ClientFloatingShip createFloatingShip(ShipType type)  
 {  
 EasyFloatingShip madeShip = (EasyFloatingShip) **shipsMade**.get(type);  
 **if** (madeShip != **null**){  
 EasyFloatingShip clone = (EasyFloatingShip) madeShip.makeCopy();  
 **return** clone;  
 }**else** {  
 ShipParameters shipParameters = **this**.getShipParameters(type);  
 EasyFloatingShip easyFloatingShip = **new** EasyFloatingShip(type);  
 setShipRotation(easyFloatingShip);  
  
 ShipConfigurator configurator = **floatingShipBuilder**.startEasy(easyFloatingShip);  
 **for** (**int** i = 0; i < shipParameters.**shipLength**; i++) {  
 configurator.addShipPart();  
 }  
  
 **if** (shipParameters.**shield**){  
 configurator.addShipShield();  
 }  
  
 madeShip = (EasyFloatingShip) configurator.build();  
 ClientShip madeShipClone = madeShip.makeCopy();  
 **shipsMade**.put(type, madeShipClone);  
  
 **return** madeShip;  
 }  
 }  
   
 **public** ClientSubmarine createSubmarine()  
 {  
 EasySubmarine madeShip = (EasySubmarine) **shipsMade**.get(ShipType.***SUBMARINE***);  
 **if** (madeShip != **null**){  
 EasySubmarine clone = (EasySubmarine) madeShip.makeCopy();  
 **return** clone;  
 }**else** {  
 ShipParameters shipParameters = **this**.getShipParameters(ShipType.***SUBMARINE***);  
 EasySubmarine easySubmarine = **new** EasySubmarine();  
 setShipRotation(easySubmarine);  
  
 ShipConfigurator configurator = **submarineBuilder**.startEasy(easySubmarine);  
 **for** (**int** i = 0; i < shipParameters.**shipLength**; i++) {  
 configurator.addShipPart();  
 }  
  
 **if** (shipParameters.**shield**){  
 configurator.addShipShield();  
 }  
  
 madeShip = (EasySubmarine) configurator.build();  
 **shipsMade**.put(ShipType.***SUBMARINE***, madeShip.makeCopy());  
  
 **return** madeShip;  
 }  
 }  
  
 **private void** setShipParameters(){  
 **for** (ShipType type: GameConstants.***shipTypes***) {  
 **this**.**shipParameters**.put(type, GameConstants.EasyLevelShips.*getShipParameters*(type));  
 }  
 }  
  
 @Override  
 **public** ShipParameters getShipParameters(ShipType type) {  
 **return** (ShipParameters) **this**.**shipParameters**.get(type);  
 }  
}

**public class** HardFloatingShip **extends** ClientFloatingShip  
{  
 **public** HardFloatingShip(ShipType type) {  
 **super**(type);  
 }  
}

**public class** MediumFloatingShip **extends** ClientFloatingShip  
{  
 **public** MediumFloatingShip(ShipType type) {  
 **super**(type);  
 }  
}

**public class** EasyFloatingShip **extends** ClientFloatingShip  
{  
 **public** EasyFloatingShip(ShipType type) {  
 **super**(type);  
 }  
}

**public class** HardSubmarine **extends** ClientSubmarine  
{  
 **public** HardSubmarine() {  
 **super**();  
 }  
}

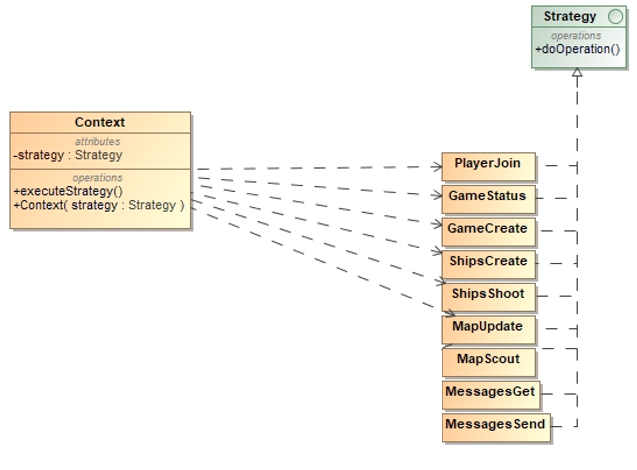
**public class** MediumSubmarine **extends** ClientSubmarine  
{  
 **public** MediumSubmarine() {  
 **super**();  
 }  
}

**public class** EasySubmarine **extends** ClientSubmarine  
{  
 **public** EasySubmarine() {  
 **super**();  
 }  
}

## Strategy

Panaudoję šį šabloną galime pagražinti kodą, kadangi atskiros funkcijos yra skirstomos į klases (strategijas). Taip pat, šio šablono pagalbą galime nesunkiai keisti algoritmus. Realizuota daugiau nei 4 strategijos klasės, kaip to ir reikalaujama.

Pav. 5 strategy šablono diagramos fragmentas



**public class** Context {  
 **private** Strategy **strategy**;  
  
 **public** Context(Strategy strategy) {  
 **this**.**strategy** = strategy;  
 }  
  
 **public** JSONObject executeStrategy(JSONObject json)  
 {  
 **return this**.**strategy**.doOperation(json);  
 }  
}

**public interface** Strategy {  
 JSONObject doOperation(JSONObject json);  
}

**public class** GameCreate **implements** Strategy {  
 @Override**public** JSONObject doOperation(JSONObject json)  
 {  
 EntityMaker entityMaker = **new** EntityMaker();  
  
 **return** entityMaker.createGame(json);  
 }  
}

**public class** GameStatus **implements** Strategy {  
 @Override  
 **public** JSONObject doOperation(JSONObject json) {  
 EntityMaker entityMaker = **new** EntityMaker();  
 Game game = **new** Game();  
  
 **return** entityMaker.getGameStatus(json, game);  
 }  
}

**public class** MapScout **implements** Strategy {  
 @Override  
 **public** JSONObject doOperation(JSONObject json) {  
 MapService mapService = **new** MapService();  
  
 **return** mapService.scoutEnemy(json);  
 }  
}

**public class** MapUpdate **implements** Strategy {  
 @Override  
 **public** JSONObject doOperation(JSONObject json) {  
 MapService mapService = **new** MapService();  
  
 **return** mapService.getOtherPlayerShootsMapUpdate(json);  
 }  
}

**public class** MessagesGet **implements** Strategy {  
 @Override  
 **public** JSONObject doOperation(JSONObject json) {  
 EntityMaker entityMaker = **new** EntityMaker();  
  
 **return** entityMaker.getMessages(json);  
 }  
}

**public class** MessagesSend **implements** Strategy {  
 @Override  
 **public** JSONObject doOperation(JSONObject json) {  
 EntityMaker entityMaker = **new** EntityMaker();;  
  
 **return** entityMaker.createMessage(json);  
 }  
}

**public class** PlayerJoin **implements** Strategy {@Override  
 **public** JSONObject doOperation(JSONObject json) {  
 EntityMaker entityMaker = **new** EntityMaker();  
  
 **return** entityMaker.playerJoin(json);  
 }  
}

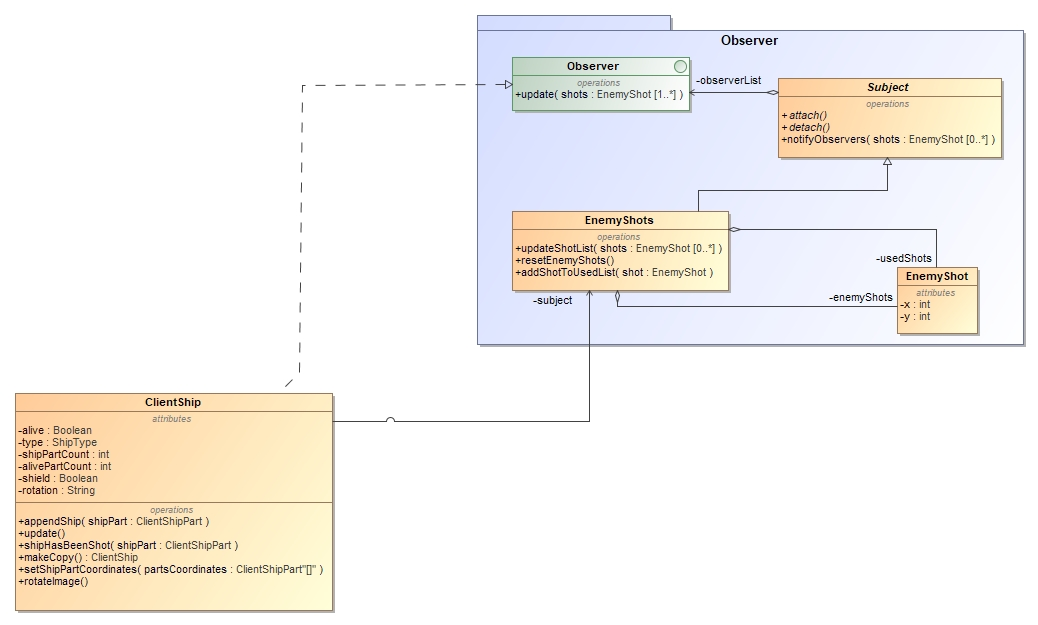
**public class** ShipsCreate **implements** Strategy {@Override  
 **public** JSONObject doOperation(JSONObject json) {  
 EntityMaker entityMaker = **new** EntityMaker();  
 **return** entityMaker.createShips(json);  
 }  
}

**public class** ShipsShoot **implements** Strategy {  
 @Override  
 **public** JSONObject doOperation(JSONObject json) {  
 EntityMaker entityMaker = **new** EntityMaker();  
 **return** entityMaker.shootShip(json);  
 }  
}

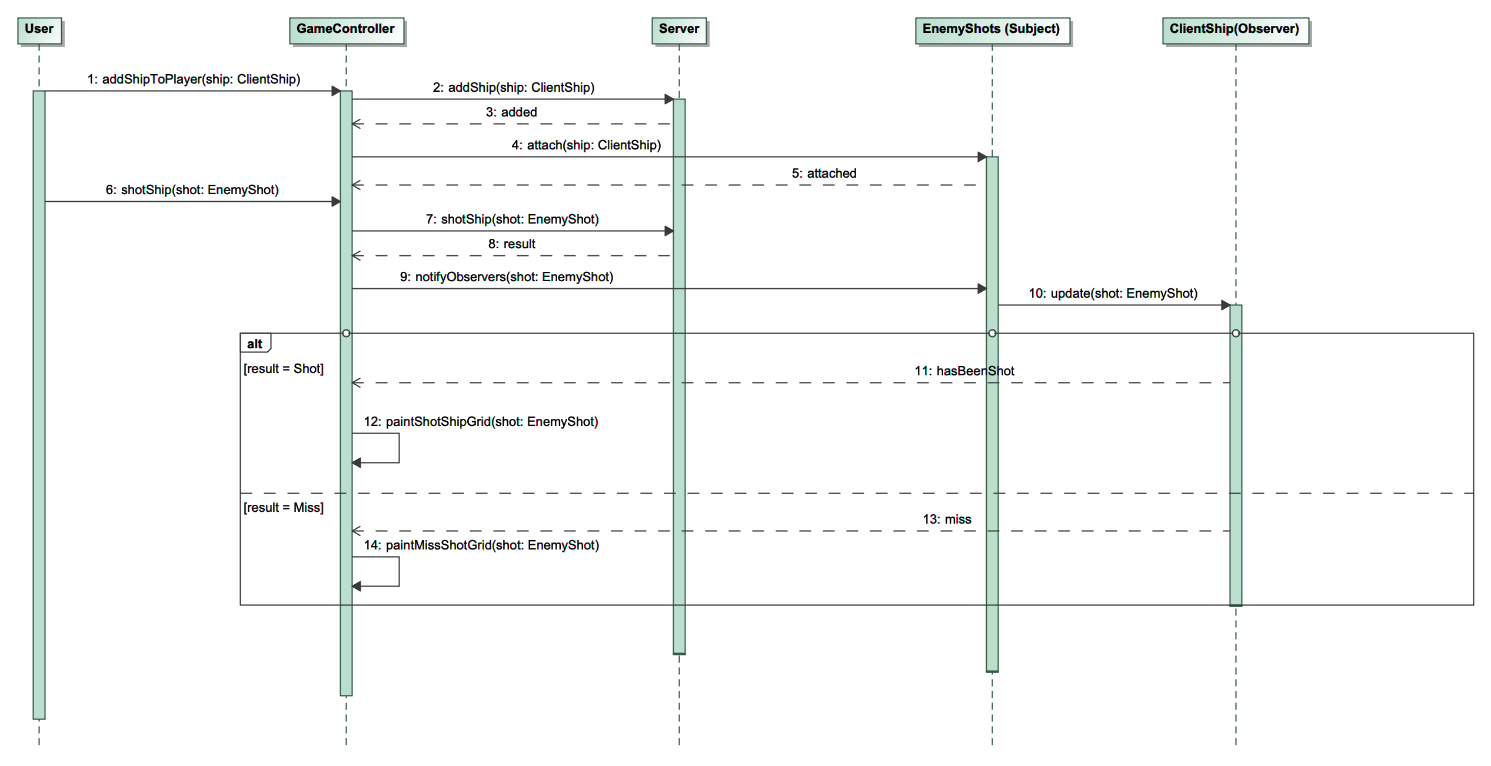
## Observer

Observer šabloną pritaikėme žemėlapio atnaujinimui, po kiekvieno raundo atnaujinamas kiekvienas laivas ir langelis. Nušovus laivą laivas iš sąrašo yra pašalintas.

Pav. 6 observer šablono diagramos fragmentas



pav. 7 observer šablono sekų diagrama



Naudojant sekų diagrama yra parodyta supaprastintas observerio veikimas.

**package** client.Models.Observer;  
  
**public class** EnemyShot  
{  
 **private int x**;  
 **private int y**;  
 **private** String **type**;  
  
 **public** EnemyShot(**int** x, **int** y, String type){  
 **this**.**x** = x;  
 **this**.**y** = y;  
 **this**.**type** = type;  
 }  
  
 **public int** getX() {  
 **return x**;  
 }  
  
 **public void** setX(**int** x) {  
 **this**.**x** = x;  
 }  
  
 **public int** getY() {  
 **return y**;  
 }  
  
 **public void** setY(**int** y) {  
 **this**.**y** = y;  
 }  
  
 **public** String getShotType(){ **return this**.**type**; }  
  
 **public void** setShotType(String type) { **this**.**type** = type; }  
  
 @Override  
 **public int** hashCode() {  
 **return super**.hashCode();  
 }  
  
 @Override  
 **public boolean** equals(Object obj) {  
 **return** (**this**.**x** == EnemyShot.**class**.cast(obj).**x** ) && (**this**.**y** == EnemyShot.**class**.cast(obj).**y**) && (**this**.**type**.equals(EnemyShot.**class**.cast(obj).**type**));  
 }  
}

**package** client.Models.Observer;  
  
**import** java.util.ArrayList;  
  
**public class** EnemyShots **extends** Subject  
{  
 **private** ArrayList<EnemyShot> **enemyShots**;  
 **private** ArrayList<EnemyShot> **usedShots**;  
  
 **public** EnemyShots() {  
 **usedShots** = **new** ArrayList<>();  
 **enemyShots** = **new** ArrayList<>();  
 }  
  
 **public void** addShotToUsedList(EnemyShot shot){  
 **usedShots**.add(shot);  
 }  
  
 **public void** resetEnemyShots(){  
 **enemyShots** = **new** ArrayList<>();  
 }  
  
 **public void** updateShotList(ArrayList<EnemyShot> shots )  
 {  
 **for** (EnemyShot shot: shots) {  
 **if** (!**this**.**usedShots**.contains(shot)){  
 **this**.**enemyShots**.add(shot);  
 }  
 }  
  
 **this**.notifyObservers(**this**.**enemyShots**);  
 }  
  
 **public** ArrayList<EnemyShot> getEnemyShots() {  
 **return enemyShots**;  
 }  
  
 **public void** notifyObservers(ArrayList<EnemyShot> shots )  
 {  
 **for** (Observer observer: **observerList**) {  
 observer.update(shots);  
 }  
 }  
  
 @Override  
 **public** Object clone() **throws** CloneNotSupportedException {  
 **return super**.clone();  
 }  
  
  
}

**package** client.Models.Observer;  
  
**import** java.util.ArrayList;  
  
**public interface** Observer  
{  
 **void** update(ArrayList<EnemyShot> shots);  
   
   
}

**package** client.Models.Observer;  
  
**import** java.util.ArrayList;  
  
**public abstract class** Subject **implements** Cloneable  
{  
 **protected** ArrayList<Observer> **observerList** = **new** ArrayList<>();  
   
 **public void** attach( Observer observer ){  
 **observerList**.add(observer);  
 }  
   
 **public void** detach( Observer observer ){  
 **observerList**.remove(observer);  
 }  
   
 **public abstract void** notifyObservers(ArrayList<EnemyShot> shots );  
   
   
}

ClientShip klasės kodas yra įdėtas ne visas, yra įdėta tik reikiama dalis observeriui.

**public class** ClientShip **implements** Observer, Cloneable  
{  
 **protected** ArrayList<ClientShipPart> **shipParts**;  
 **protected** EnemyShots **subject**;  
   
 **private** Boolean **alive**;  
 **private** ShipType **type**;  
 **private int shipPartCount**;  
 **private int alivePartCount**;  
 **private** String **rotation**;  
 **private boolean shield** = **false**;  
  
 **public** ClientShip(){  
 **this**.**shipPartCount** = 0;  
 **this**.**alivePartCount** = 0;  
 **shipParts** = **new** ArrayList<>();  
 }

**public** EnemyShots getSubject() {  
 **return subject**;  
 }  
  
 **public void** setSubject(EnemyShots subject) {  
 **this**.**subject** = subject;  
 }  
  
 **public int** getAlivePartCount() {  
 **return alivePartCount**;  
 }  
  
 **public void** setAlivePartCount(**int** alivePartCount) {  
 **this**.**alivePartCount** = alivePartCount;  
 }  
  
 **public** ArrayList<ClientShipPart> getShipParts() {  
 **return this**.**shipParts**;  
}  
  
**public** Boolean getAlive() {  
 **return alive**;  
}  
  
**public** ShipType getType() {  
 **return type**;  
}  
  
**public void** setAlive(Boolean alive) {  
 **this**.**alive** = alive;  
}  
  
**public void** setType(ShipType type) {  
 **this**.**type** = type;  
}  
  
**public int** getShipPartCount() {  
 **return shipPartCount**;  
}  
  
**public void** setShipPartCount(**int** shipPartCount) {  
 **this**.**shipPartCount** = shipPartCount;  
}  
  
**public void** appendShip(ClientShipPart shipPart){  
 **this**.**shipParts**.add(shipPart);  
 **alivePartCount**++;  
 **shipPartCount**++;  
}  
  
 *// Check yourself* **private void** shipHasBeenShot(ClientShipPartInterface shipPart){  
 GameController.*playerShipHasBeenShot*(shipPart);  
 }

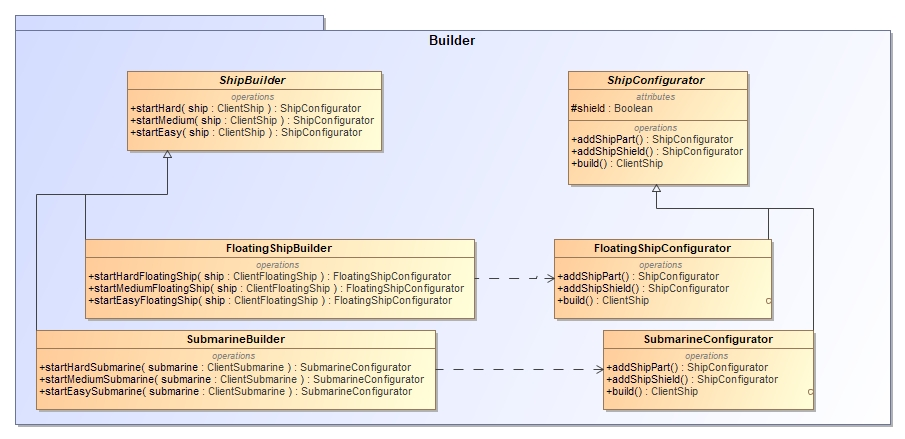
@Override  
 **public void** update(ArrayList<EnemyShot> shots) {  
  
 ArrayList<EnemyShot> shipHits = **new** ArrayList<>();  
 *// Check if my ships ok or no :D* **int** count = 0;  
 **for** (ClientShipPart shipPart: **shipParts**) {  
  
 **for** (EnemyShot shot: shots) {  
 *// Ship part has been shot* **if** (shipPart.getX() == shot.getX() && shipPart.getY() == shot.getY() && shipPart.getPart().getShipType() != ShipType.***SUBMARINE***) {  
  
 **if**(shipPart.getHitOnce() && **shield**) {  
 **alivePartCount**--;  
 shipPart.setColor(GameConstants.***hitShipColor***);  
 shipPart.setHit(**true**);  
  
 ClientShipPartDamaged dmgPart = **new** ClientShipPartDamaged(shipPart);  
 **subject**.addShotToUsedList(shot);  
  
 *// Painting* shipHasBeenShot(dmgPart);  
  
 shipHits.add(shot);  
  
 *// Checking if ship has died* **if** (**alivePartCount** <= 0) {  
 **alive** = **false**;  
 **break**;  
 }  
 }  
 **else if**(!shipPart.getHitOnce() && **shield**) {  
  
 ArrayList<ClientShipPart> parts = **this**.getShipParts();  
 ShipParameters shipParameters = GameData.*shipAbstractFactory*.getShipParameters(**this**.getType());  
 ClientShipPart partWithoutArmour = **new** ClientShipPart();  
 partWithoutArmour.setCoordinates(shipPart.getX(), shipPart.getY());  
 partWithoutArmour.setAngle(shipPart.getAngle());  
 **if**(count == 0) {  
 *// Painting ship nose* ClientShipPartInterface paintedPart = **new** ClientShipPartBaseShip(**new** ClientShipPartBase(partWithoutArmour), shipParameters.**shipImages**[0]);  
  
 partWithoutArmour.setPartImage(paintedPart.getImage());  
*// RotateImage(paintedPart);* shipHasBeenShot(paintedPart);  
 }  
 **else if**(count < parts.size()-1)  
 {  
 ClientShipPartInterface paintedPart = **new** ClientShipPartBaseShip(**new** ClientShipPartBase(partWithoutArmour), shipParameters.**shipImages**[1]);  
 partWithoutArmour.setPartImage(paintedPart.getImage());  
*// RotateImage(paintedPart);* shipHasBeenShot(paintedPart);  
 }  
 **else if**(count == parts.size()-1)  
 {  
 ClientShipPartInterface paintedPart = **new** ClientShipPartBaseShip(**new** ClientShipPartBase(partWithoutArmour), shipParameters.**shipImages**[shipParameters.**shipImages**.**length**-1]);  
 partWithoutArmour.setPartImage(paintedPart.getImage());  
*// RotateImage(paintedPart);* shipHasBeenShot(paintedPart);  
 }  
  
 shipPart = partWithoutArmour;  
 shipHits.add(shot);  
 shipPart.setHitOnce(**true**);  
  
 }  
 **else if**(!**shield**) {  
  
 **alivePartCount**--;  
 shipPart.setColor(GameConstants.***hitShipColor***);  
 shipPart.setHit(**true**);  
  
 ClientShipPartDamaged dmgPart = **new** ClientShipPartDamaged(shipPart);  
 **subject**.addShotToUsedList(shot);  
  
 *// Painting* shipHasBeenShot(dmgPart);  
  
 shipHits.add(shot);  
  
 *// Checking if ship has died* **if** (**alivePartCount** <= 0) {  
 **alive** = **false**;  
 **break**;  
 }  
  
 }  
 }  
 **else if** (shipPart.getX() == shot.getX() && shipPart.getY() == shot.getY() && shipPart.getPart().getShipType() == ShipType.***SUBMARINE*** && shot.getShotType().equals(GameConstants.***simpleShoot***)) {  
  
 shipHits.add(shot);  
  
 }  
 **else if**(shipPart.getX() == shot.getX() && shipPart.getY() == shot.getY() && shipPart.getPart().getShipType() == ShipType.***SUBMARINE*** && ( shot.getShotType().equals(GameConstants.***superShoot***) || shot.getShotType().equals(GameConstants.***underWaterShoot***) ))  
 {  
 **if**(shipPart.getHitOnce() && **shield**) {  
 **alivePartCount**--;  
 shipPart.setColor(GameConstants.***hitShipColor***);  
 shipPart.setHit(**true**);  
  
 ClientShipPartDamaged dmgPart = **new** ClientShipPartDamaged(shipPart);  
 **subject**.addShotToUsedList(shot);  
  
 *// Painting* shipHasBeenShot(dmgPart);  
  
 shipHits.add(shot);  
  
 *// Checking if ship has died* **if** (**alivePartCount** <= 0) {  
 **alive** = **false**;  
 **break**;  
 }  
 }  
 **else if**(!shipPart.getHitOnce() && **shield**) {  
  
 ArrayList<ClientShipPart> parts = **this**.getShipParts();  
 ShipParameters shipParameters = GameData.*shipAbstractFactory*.getShipParameters(**this**.getType());  
 ClientShipPart partWithoutArmour = **new** ClientShipPart();  
 partWithoutArmour.setCoordinates(shipPart.getX(), shipPart.getY());  
 partWithoutArmour.setAngle(shipPart.getAngle());  
 **if**(count == 0) {  
 *// Painting ship nose* ClientShipPartInterface paintedPart = **new** ClientShipPartBaseShip(**new** ClientShipPartBase(partWithoutArmour), shipParameters.**shipImages**[0]);  
  
 partWithoutArmour.setPartImage(paintedPart.getImage());  
*// RotateImage(paintedPart);* shipHasBeenShot(paintedPart);  
 }  
 **else if**(count < parts.size()-1)  
 {  
 ClientShipPartInterface paintedPart = **new** ClientShipPartBaseShip(**new** ClientShipPartBase(partWithoutArmour), shipParameters.**shipImages**[1]);  
 partWithoutArmour.setPartImage(paintedPart.getImage());  
*// RotateImage(paintedPart);* shipHasBeenShot(paintedPart);  
 }  
 **else if**(count == parts.size()-1)  
 {  
 ClientShipPartInterface paintedPart = **new** ClientShipPartBaseShip(**new** ClientShipPartBase(partWithoutArmour), shipParameters.**shipImages**[shipParameters.**shipImages**.**length**-1]);  
 partWithoutArmour.setPartImage(paintedPart.getImage());  
*// RotateImage(paintedPart);* shipHasBeenShot(paintedPart);  
 }  
  
 shipPart = partWithoutArmour;  
 shipHits.add(shot);  
 shipPart.setHitOnce(**true**);  
  
 }  
 **else if**(!**shield**) {  
  
 **alivePartCount**--;  
 shipPart.setColor(GameConstants.***hitShipColor***);  
 shipPart.setHit(**true**);  
  
 ClientShipPartDamaged dmgPart = **new** ClientShipPartDamaged(shipPart);  
 **subject**.addShotToUsedList(shot);  
  
 *// Painting* shipHasBeenShot(dmgPart);  
  
 shipHits.add(shot);  
  
 *// Checking if ship has died* **if** (**alivePartCount** <= 0) {  
 **alive** = **false**;  
 **break**;  
 }  
  
 }  
 }  
  
 }  
  
 *// If ship died, detach from Subject as Observer* **if** (!**alive**){  
*// subject.detach(this);* **break**;  
 }  
 count++;  
 }  
  
 *//* ***TODO: [Check For Errors]*** shots.removeAll(shipHits);  
 }

}

## Builder

Šį šabloną pritaikėme laivų gaminimui, kas leidžia geriau valdyti laivų kūrimo procesą.

Pav. 8 builder šablono kodo fragmentas



Builder yra sujungtas su AbstractFactory. AbstractFactory naudoja Builder, kad sudėliotų tam tikrą laivą pagal tam tikrą žaidimo lygį iš tam tikrų galimų detalių pvz. (šarvų pridėjimas tam tikram laivui).

**public abstract class** ShipBuilder {  
 ClientShip **ship**;  
  
 **public abstract** ShipConfigurator startHard(ClientShip ship);  
 **public abstract** ShipConfigurator startMedium(ClientShip ship);  
 **public abstract** ShipConfigurator startEasy(ClientShip ship);  
}

**public class** FloatingShipBuilder **extends** ShipBuilder  
{  
 **public** FloatingShipBuilder(){  
 }  
   
 **public** ShipConfigurator startHard( ClientShip ship )  
 {  
 **this**.**ship** = ship;  
 **return new** FloatingShipConfigurator(ship);  
 }  
   
 **public** ShipConfigurator startMedium( ClientShip ship )  
 {  
 **this**.**ship** = ship;  
 **return new** FloatingShipConfigurator(ship);  
 }  
   
 **public** ShipConfigurator startEasy( ClientShip ship )  
 {  
 **this**.**ship** = ship;  
 **return new** FloatingShipConfigurator(ship);  
 }  
}

**public class** SubmarineBuilder **extends** ShipBuilder  
{  
 **public** SubmarineBuilder(){}  
  
 **public** ShipConfigurator startHard( ClientShip ship )  
 {  
 **this**.**ship** = ship;  
 **return new** SubmarineConfigurator(ship);  
 }  
  
 **public** ShipConfigurator startMedium( ClientShip ship )  
 {  
 **this**.**ship** = ship;  
 **return new** SubmarineConfigurator(ship);  
 }  
  
 **public** ShipConfigurator startEasy( ClientShip ship )  
 {  
 **this**.**ship** = ship;  
 **return new** SubmarineConfigurator(ship);  
 }  
}

**public abstract class** ShipConfigurator {  
 **protected** ClientShip **ship**;  
 **protected boolean shield** = **false**;  
  
 **public abstract** ShipConfigurator addShipPart();  
 **public abstract** ShipConfigurator addShipShield();  
 **public abstract** ClientShip build();  
}

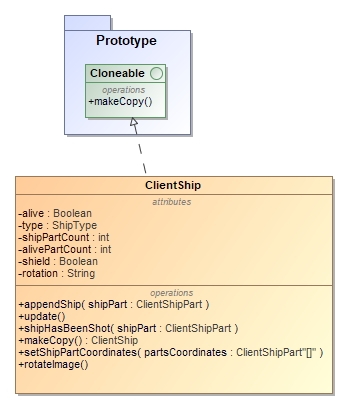
**public class** FloatingShipConfigurator **extends** ShipConfigurator  
{  
 **private boolean shield** = **false**;  
  
 **public** FloatingShipConfigurator(ClientShip floatingShip){  
 **this**.**ship** = floatingShip;  
 }  
   
 **public** FloatingShipConfigurator addShipPart( )  
 {  
 ClientShipPart shipPart = **new** ClientShipPart();  
 **ship**.appendShip(shipPart);  
 **return this**;  
 }  
   
 **public** FloatingShipConfigurator addShipShield( )  
 {  
 **shield** = **true**;  
 **return this**;  
 }  
   
 **public** ClientFloatingShip build( )  
 {  
 applyDefaultSkinToShip();  
 **if** (**shield**){  
 applyShieldSkinToShip();  
 **ship**.setShield(**true**);  
 }  
 **return** (ClientFloatingShip) **ship**;  
 }  
  
 **public void** applyShieldSkinToShip(){  
 ArrayList<ClientShipPart> parts = **ship**.getShipParts();  
  
 **for** (**int** i = 0; i < **ship**.getShipPartCount(); i++) {  
 ClientShipPart shipPart = parts.get(i);  
 ClientShipPartInterface paintedPart = **new** ClientShipPartArmour(shipPart);  
  
 shipPart.setPartImage(paintedPart.getImage());  
 **ship**.setShipPart(i, shipPart);  
 }  
 }  
  
 **public void** applyDefaultSkinToShip(){  
 ArrayList<ClientShipPart> parts = **ship**.getShipParts();  
 ShipParameters shipParameters = GameData.*shipAbstractFactory*.getShipParameters(**ship**.getType());  
  
 *// Painting ship nose* ClientShipPart shipPart = parts.get(0);  
 ClientShipPartInterface paintedPart = **new** ClientShipPartBaseShip( **new** ClientShipPartBase(shipPart), shipParameters.**shipImages**[0] );  
  
 shipPart.setPartImage(paintedPart.getImage());  
 **ship**.setShipPart(0, shipPart);  
  
 *// Painting ship body* **for** (**int** i = 1; i < parts.size()-1; i++) {  
 shipPart = parts.get(i);  
 paintedPart = **new** ClientShipPartBaseShip( **new** ClientShipPartBase(shipPart), shipParameters.**shipImages**[1] );  
  
 shipPart.setPartImage(paintedPart.getImage());  
 **ship**.setShipPart(i, shipPart);  
 }  
  
 *// Painting ship tail* **if** (**ship**.getShipPartCount() > 1){  
 shipPart = parts.get(parts.size()-1);  
 paintedPart = **new** ClientShipPartBaseShip( **new** ClientShipPartBase(shipPart), shipParameters.**shipImages**[shipParameters.**shipImages**.**length**-1] );  
  
 shipPart.setPartImage(paintedPart.getImage());  
 **ship**.setShipPart(parts.size()-1, shipPart);  
 }  
  
 }  
}

**public class** SubmarineConfigurator **extends** ShipConfigurator  
{  
 **public** SubmarineConfigurator(ClientShip submarine){  
 **this**.**ship** = submarine;  
 }  
   
 **public** SubmarineConfigurator addShipPart( )  
 {  
 ClientShipPart shipPart = **new** ClientShipPart();  
 **ship**.appendShip(shipPart);  
 **return this**;  
 }  
   
 **public** SubmarineConfigurator addShipShield( )  
 {  
 **shield** = **true**;  
 **return this**;  
 }  
  
 **public** ClientSubmarine build( )  
 {  
 applyDefaultSkinToShip();  
 **if** (**shield**){  
 applyShieldSkinToShip();  
 **ship**.setShield(**true**);  
 }  
 **return** (ClientSubmarine) **ship**;  
 }  
  
 **public void** applyShieldSkinToShip(){  
 ArrayList<ClientShipPart> parts = **ship**.getShipParts();  
  
 **for** (**int** i = 0; i < **ship**.getShipPartCount(); i++) {  
 ClientShipPart shipPart = parts.get(i);  
 ClientShipPartInterface paintedPart = **new** ClientShipPartArmour(shipPart);  
  
 shipPart.setPartImage(paintedPart.getImage());  
 **ship**.setShipPart(i, shipPart);  
 }  
 }  
  
 **public void** applyDefaultSkinToShip(){  
 ArrayList<ClientShipPart> parts = **ship**.getShipParts();  
 ShipParameters shipParameters = GameData.*shipAbstractFactory*.getShipParameters(**ship**.getType());  
  
 *// Painting ship nose* ClientShipPart shipPart = parts.get(0);  
 ClientShipPartInterface paintedPart = **new** ClientShipPartBaseShip( **new** ClientShipPartBase(shipPart), shipParameters.**shipImages**[0] );  
  
 shipPart.setPartImage(paintedPart.getImage());  
 **ship**.setShipPart(0, shipPart);  
  
 *// Painting ship body* **for** (**int** i = 1; i < parts.size()-1; i++) {  
 shipPart = parts.get(i);  
 paintedPart = **new** ClientShipPartBaseShip( **new** ClientShipPartBase(shipPart), shipParameters.**shipImages**[1] );  
  
 shipPart.setPartImage(paintedPart.getImage());  
 **ship**.setShipPart(i, shipPart);  
 }  
  
 *// Painting ship tail* **if** (**ship**.getShipPartCount() > 1){  
 shipPart = parts.get(parts.size()-1);  
 paintedPart = **new** ClientShipPartBaseShip( **new** ClientShipPartBase(shipPart), shipParameters.**shipImages**[shipParameters.**shipImages**.**length**-1] );  
  
 shipPart.setPartImage(paintedPart.getImage());  
 **ship**.setShipPart(parts.size()-1, shipPart);  
 }  
 }  
}

## Prototype

Šis šablonas paslepia objektų kūrimo komplikacijas. Taip pat sumažina reikalą kurti sub klases. Prototype yra realizuojamas per konstruktorių, nes nėra pasiekiamos visos naudojamos klasės, kadangi ne visas jas galime modifikuoti. AbtractFactory šablonas sukuria laivą, tačiau jei yra prašomas toks pat laivo tipas, yra grąžinama kopija.

Pav. 9 prototype šablono diagramos fragmentas



Šio šablono programos kodas:

**public class** ClientShip **implements** Observer, Cloneable  
{

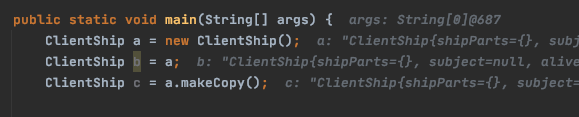
**public** ClientShip makeCopy(){  
 **try** {  
 ClientShip clone = (ClientShip) **this**.clone();  
  
 clone.**shipParts** = **new** ArrayList<ClientShipPart>();  
 **for** (ClientShipPart part: **this**.**shipParts**) {  
 clone.**shipParts**.add((ClientShipPart) part.clone());  
 }**return** clone;  
 }**catch** (CloneNotSupportedException ex){  
 ex.printStackTrace();  
 **return this**;  
 }  
 }

}

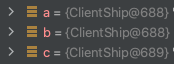
@Override  
**public** Object clone() **throws** CloneNotSupportedException {  
 **return super**.clone();  
}

Šablono realizavimo pavyzdys:

Pav. 10 prototype šablono pavyzdys



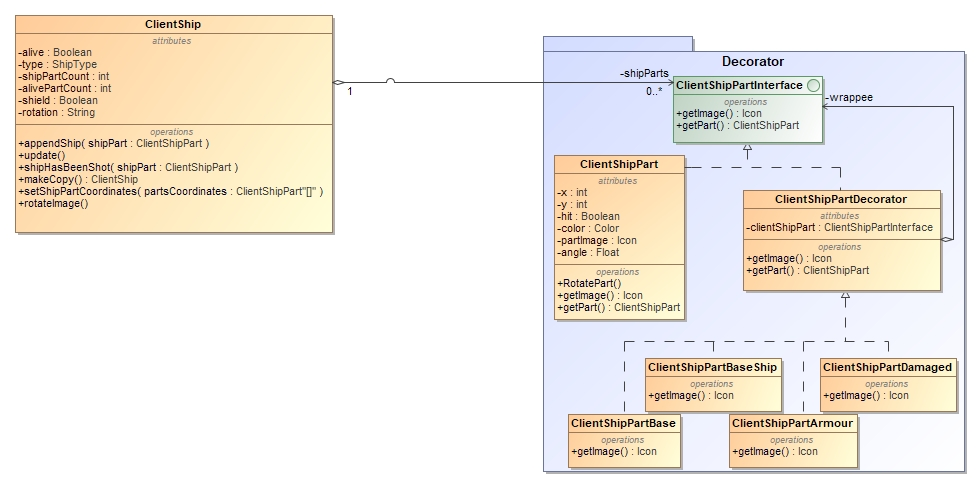
Pav. 11 objektų numeriai



## Decorator

Šį šabloną panaudojome lengvesniam laivo dalių dekoravimui, t.y. šio šablono pagalba nesunkiai laivo daliai galime uždėti bazę, šarvus, pašauto laivo išvaizdą, bei dabar nesunkiai galime pridėti laivo dalies dekoravimo komponentus, jei ateityje sugalvotume pridėti tarkim naujas patrankas. Tačiau yra sudėtinga panaikinti tam tikrą konkretų dekoratoriaus sluoksnį.

Pav. 12 decorator šablono diagramos fragmentas



Yra keturi dekoravimo lygiai – bazė, laivo bazė, laivo šarvai, bei pašautas laivas. Šio šablono kodas:

**package** client.Models.Decorator;  
  
**import** client.GameConstants.ComponentsSizeConstants;  
**import** client.GameConstants.GameConstants;  
**import** client.Models.GameData;  
  
**import** javax.imageio.ImageIO;  
**import** javax.swing.\*;  
**import** java.awt.\*;  
**import** java.io.FileInputStream;  
  
**public class** ClientShipPartArmour **extends** ClientShipPartDecorator {  
 **private** Image **shipPartImage** = **null**;  
  
 **public** ClientShipPartArmour(ClientShipPartInterface decoratedShipPart) {  
 **super**(decoratedShipPart);  
 **try** {  
 **shipPartImage** = ImageIO.*read*(**new** FileInputStream(GameConstants.*shipShieldImage*));  
 **if** (**shipPartImage** != **null**) {  
 **shipPartImage** = **shipPartImage**.getScaledInstance(ComponentsSizeConstants.***gridButtonWidth***, ComponentsSizeConstants.***gridButtonHeight***, Image.***SCALE\_DEFAULT***);  
 }  
 } **catch** (Exception e) {  
 System.***out***.println(e.toString());  
 }  
 }  
  
 @Override  
 **public** Icon getImage() {  
 **return new** MergedIcon(**clientShipPart**.getImage(), **new** ImageIcon(**shipPartImage**));  
 }  
}

**package** client.Models.Decorator;  
  
**import** client.GameConstants.ComponentsSizeConstants;  
**import** client.GameConstants.GameConstants;  
**import** client.Models.GameData;  
  
**import** javax.swing.\*;  
**import** java.awt.\*;  
**import** java.awt.image.BufferedImage;  
  
**import static** client.Models.Decorator.ImageUtils.*rotateImage*;  
  
**public class** ClientShipPartBase **extends** ClientShipPartDecorator {  
 **private** BufferedImage **newImage** = **new** BufferedImage(ComponentsSizeConstants.***gridButtonWidth***, ComponentsSizeConstants.***gridButtonHeight***, BufferedImage.***TYPE\_INT\_ARGB***);  
  
 **public** ClientShipPartBase(ClientShipPartInterface decoratedShipPart) {  
 **super**(decoratedShipPart);  
 Graphics2D graphics = **newImage**.createGraphics();  
 graphics.setPaint(**new** Color(59, 222, 0));  
 graphics.fillRect(0,0, **newImage**.getWidth(), **newImage**.getHeight());  
 }  
  
 @Override  
 **public** Icon getImage() {  
 **return new** MergedIcon(**new** ImageIcon(**newImage**), **clientShipPart**.getImage());  
 }  
  
  
  
}

**package** client.Models.Decorator;  
  
**import** client.GameConstants.ComponentsSizeConstants;  
**import** client.GameConstants.GameConstants;  
**import** client.Models.GameData;  
  
**import** javax.imageio.ImageIO;  
**import** javax.swing.\*;  
**import** java.awt.\*;  
**import** java.awt.image.BufferedImage;  
**import** java.io.FileInputStream;  
  
**import static** client.Models.Decorator.ImageUtils.\*;  
  
**public class** ClientShipPartBaseShip **extends** ClientShipPartDecorator {  
 **private** Image **shipPartImage** = **null**;  
  
 **public** ClientShipPartBaseShip(ClientShipPartInterface decoratedShipPart, String image) {  
 **super**(decoratedShipPart);  
 **try** {  
  
 **shipPartImage** = *rotateImage*(ImageIO.*read*( **new** FileInputStream(image) ), -90);  
  
 Icon oldIcon = **new** ImageIcon(**shipPartImage**);  
 BufferedImage bi = **new** BufferedImage(  
 oldIcon.getIconWidth(),  
 oldIcon.getIconHeight(),  
 BufferedImage.***TYPE\_INT\_ARGB***);  
 Graphics g = bi.createGraphics();  
 oldIcon.paintIcon(**null**, g, 0,0);  
 g.dispose();  
 **shipPartImage** = *rotateImage*(bi, **this**.getPart().getAngle());  
  
 **shipPartImage** = **shipPartImage**.getScaledInstance(ComponentsSizeConstants.***gridButtonWidth***, ComponentsSizeConstants.***gridButtonHeight***, Image.***SCALE\_DEFAULT***);  
 **if** (**shipPartImage** != **null**) {  
 **shipPartImage** = **shipPartImage**.getScaledInstance(ComponentsSizeConstants.***gridButtonWidth***, ComponentsSizeConstants.***gridButtonHeight***, Image.***SCALE\_DEFAULT***);  
 }  
 } **catch** (Exception e) {  
 System.***out***.println(e.toString());  
 }  
 }  
  
 @Override  
 **public** Icon getImage() {  
 **return new** MergedIcon(**clientShipPart**.getImage(), **new** ImageIcon(**shipPartImage**));  
 }  
  
}

**package** client.Models.Decorator;  
  
**import** client.GameConstants.ComponentsSizeConstants;  
**import** client.GameConstants.GameConstants;  
**import** client.Models.GameData;  
  
**import** javax.imageio.ImageIO;  
**import** javax.swing.\*;  
**import** java.awt.\*;  
**import** java.awt.image.BufferedImage;  
**import** java.io.FileInputStream;  
  
**import static** client.Models.Decorator.ImageUtils.*rotateImage*;  
  
**public class** ClientShipPartDamaged **extends** ClientShipPartDecorator {  
 **private** Image **shipPartImage** = **null**;  
  
 **public** ClientShipPartDamaged(ClientShipPartInterface decoratedShipPart) {  
 **super**(decoratedShipPart);  
 **try** {  
 **shipPartImage** = *rotateImage*(ImageIO.*read*(**new** FileInputStream(GameConstants.*shipDamagedImage*)), -90);  
  
 Icon oldIcon = **new** ImageIcon(**shipPartImage**);  
 BufferedImage bi = **new** BufferedImage(  
 oldIcon.getIconWidth(),  
 oldIcon.getIconHeight(),  
 BufferedImage.***TYPE\_INT\_ARGB***);  
 Graphics g = bi.createGraphics();  
 oldIcon.paintIcon(**null**, g, 0,0);  
 g.dispose();  
 **shipPartImage** = *rotateImage*(bi, **this**.getPart().getAngle());  
  
 **if** (**shipPartImage** != **null**) {  
 **shipPartImage** = **shipPartImage**.getScaledInstance(ComponentsSizeConstants.***gridButtonWidth***, ComponentsSizeConstants.***gridButtonHeight***, Image.***SCALE\_DEFAULT***);  
 }  
 } **catch** (Exception e) {  
 System.***out***.println(e.toString());  
 }  
 }  
  
 @Override  
 **public** Icon getImage() {  
 **return new** MergedIcon(**clientShipPart**.getImage(), **new** ImageIcon(**shipPartImage**));  
 }  
}

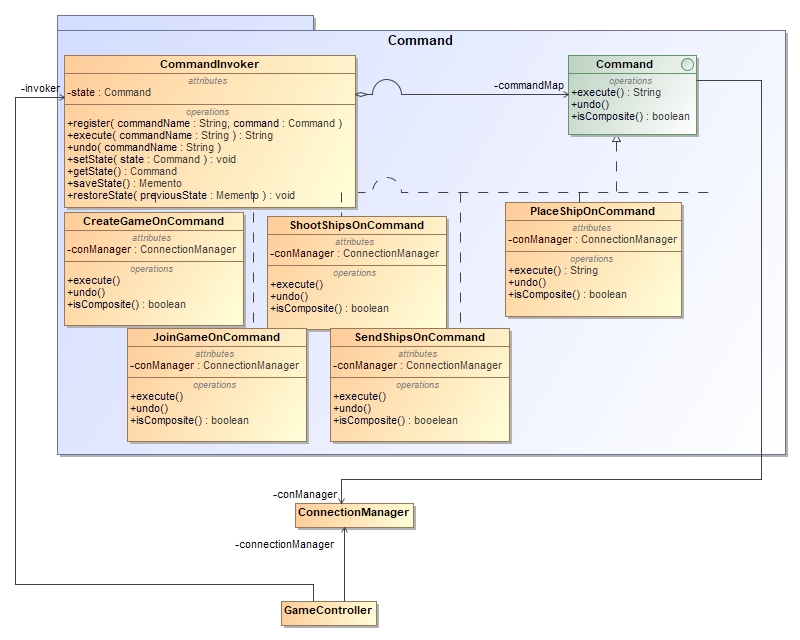
**package** client.Models.Decorator;  
  
**import** client.Models.ClientShipPart;  
  
**import** javax.swing.\*;  
  
  
**public abstract class** ClientShipPartDecorator **implements** ClientShipPartInterface {  
 **protected** ClientShipPartInterface **clientShipPart**;  
  
 **public** ClientShipPartDecorator(ClientShipPartInterface decoratedShipPart) {  
 **clientShipPart** = decoratedShipPart;  
 }  
  
 @Override  
 **public** Icon getImage() {  
 **return new** MergedIcon(**clientShipPart**.getImage(), **clientShipPart**.getImage());  
 }  
  
 @Override  
 **public** ClientShipPart getPart() {  
 **return clientShipPart**.getPart();  
 }  
}

**package** client.Models.Decorator;  
  
**import** client.Models.ClientShipPart;  
  
**import** javax.swing.\*;  
  
**public interface** ClientShipPartInterface {  
 Icon getImage();  
 ClientShipPart getPart();  
}

## Command

Pritaikėme šį šabloną, tam kad komandų paleidėjas neturi nieko žinoti apie komandų veikimą ir gali jas vykdyti. Tai leidžia nesunkiai pridėti naujų komandų, taip pat turėti sąrašą komandų, kurias galime paleisti norima arba reikiama tvarka. Undo() galimybė yra realizuota vienos komandos, t.y. PlaceShipOnCommand. Šią komandą galima atstatyti į buvusią padėtį. Į komandas kreipiasi GameController.

Pav. 13 command šablono diagramos fragmentas



Šio šablono programos kodas:

**package** client.Models.Command;  
  
**public interface** Command {  
 String execute();  
 **void** undo();  
}

**package** client.Models.Command;  
  
**import** client.Models.ConnectionManager;  
  
**public class** CreateGameOnCommand **implements** Command {  
 **private final** ConnectionManager **conManager**;  
  
 **public** CreateGameOnCommand(ConnectionManager conManager) {  
 **this**.**conManager** = conManager;  
 }  
  
 @Override  
 **public** String execute() {  
 **return conManager**.createGame();  
 }  
  
 @Override  
 **public void** undo() {  
  
 }  
}

**package** client.Models.Command;  
  
**import** client.Models.ConnectionManager;  
  
**public class** JoinGameOnCommand **implements** Command{  
 **private** String **username**;  
 **private final** ConnectionManager **conManager**;  
  
 **public** JoinGameOnCommand(String username, ConnectionManager conManager) {  
 **this**.**username** = username;  
 **this**.**conManager** = conManager;  
 }  
  
 @Override  
 **public** String execute() {  
 **return conManager**.joinGame(**username**);  
 }  
  
 @Override  
 **public void** undo() {  
  
 }  
}

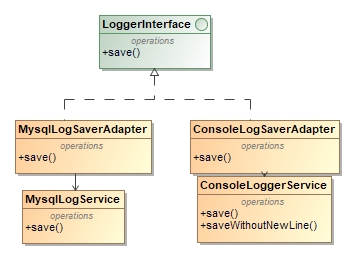
**package** client.Models.Command;  
  
**import** client.Models.ClientShip;  
**import** client.Models.ConnectionManager;  
**import** client.Models.Decorator.ClientShipPartInterface;  
**import** client.Views.SeaButton;  
**import** client.Views.ShipButton;  
  
**import** java.util.ArrayList;  
  
**public class** PlaceShipOnCommand **implements** Command {  
  
 **private** ConnectionManager **conManager**;  
 **private** SeaButton[][][] **seaButtons**;  
 **private** SeaButton **seaButton**;  
 **private** ShipButton **refShipButton**;  
 **private** ArrayList<ClientShipPartInterface> **newShipParts**;  
 **private** String **rotation**;  
 **private** ClientShip **originalShip**;  
 **private int callerGridIndex**;  
 **private int shipLength**;  
 **private int shipsLeft**;  
  
 **public** PlaceShipOnCommand(SeaButton[][][] seaButtons, SeaButton seaButton, ShipButton refShipButton, ArrayList<ClientShipPartInterface> newShipParts, String rotation, ClientShip originalShip, **int** callerGridIndex, **int** shipLength, **int** shipsLeft, ConnectionManager conManager) {  
 **this**.**seaButtons** = seaButtons;  
 **this**.**seaButton** = seaButton;  
 **this**.**refShipButton** = refShipButton;  
 **this**.**newShipParts** = newShipParts;  
 **this**.**rotation** = rotation;  
 **this**.**originalShip** = originalShip;  
 **this**.**callerGridIndex** = callerGridIndex;  
 **this**.**shipLength** = shipLength;  
 **this**.**shipsLeft** = shipsLeft;  
  
 **this**.**conManager** = conManager;  
 }  
  
 @Override  
 **public** String execute() {  
 **return conManager**.placeShip(**seaButtons**, **seaButton**, **newShipParts**, **callerGridIndex**, **shipLength**, **shipsLeft**);  
 }  
  
 @Override  
 **public void** undo() {  
 **conManager**.revertPlacement(**seaButtons**, **seaButton**, **newShipParts**, **rotation**, **callerGridIndex**, **shipLength**, **shipsLeft**);  
 }  
  
 **public** ClientShip getUndoneShip() {  
 **return originalShip**;  
 }  
  
 **public** ShipButton refShipButton() {  
 **return refShipButton**;  
 }  
}

**package** client.Models.Command;  
  
**import** client.Models.ClientShip;  
**import** client.Models.ClientShipPart;  
**import** client.Models.ConnectionManager;  
**import** org.json.JSONArray;  
**import** org.json.JSONObject;  
  
**import** java.util.ArrayList;  
  
**package** client.Models.Command;  
  
**import** client.Models.ClientShip;  
**import** client.Models.ConnectionManager;  
  
**import** java.util.ArrayList;  
  
**public class** SendShipsOnCommand **implements** Command {  
  
 **private int gameId**;  
 **private int playedId**;  
 **private** ArrayList<ClientShip> **ships**;  
 **private** ConnectionManager **conManager**;  
  
 **public** SendShipsOnCommand(**int** gameId, **int** playedId, ArrayList<ClientShip> ships, ConnectionManager conManager) {  
 **this**.**gameId** = gameId;  
 **this**.**playedId** = playedId;  
 **this**.**ships** = ships;  
 **this**.**conManager** = conManager;  
 }  
  
 @Override  
 **public** String execute() {  
 **return conManager**.sendShips(**gameId**, **playedId**, **ships**, **conManager**);  
 }  
  
 @Override  
 **public void** undo() {  
  
 }  
}

**package** client.Models.Command;  
  
**import** client.Models.ConnectionManager;  
**public class** ShootShipOnCommand **implements** Command {  
  
 **private int gameId**;  
 **private int playerId**;  
 **private** String **coordinates**;  
 **private** String **type**;  
 **private** ConnectionManager **conManager**;  
  
 **public** ShootShipOnCommand(**int** gameId, **int** playerId, String coordinates, String type, ConnectionManager connectionManager) {  
 **this**.**gameId** = gameId;  
 **this**.**playerId** = playerId;  
 **this**.**coordinates** = coordinates;  
 **this**.**type** = type;  
 **this**.**conManager** = connectionManager;  
 }  
  
 @Override  
 **public** String execute() {  
 **return conManager**.shootShip(**gameId**, **playerId**, **coordinates**, **type**, **conManager**);  
 }  
  
 @Override  
 **public void** undo() {  
  
 }  
}

## Adapter

Pav. 14 adapter šablono diagramos fragmentas



Adapteris naudojamas saugot logus į MySQL arba į Console. Adapter ir adaptee skiriasi metodų kiekis kaip ir yra reikalauta reikalavimuose.

**public interface** LoggerInterface {  
 **public void** save(JSONObject log);  
}

**public class** MysqlLogSaverAdapter **implements** LoggerInterface {  
 @Override  
 **public void** save(JSONObject log) {  
 MysqlLoggerService mysqlLoggerService = **new** MysqlLoggerService();  
  
 String gameId = log.get(**"GameId"**).toString();  
 String event = log.toString();  
  
 mysqlLoggerService.save(gameId, event);  
 }  
}

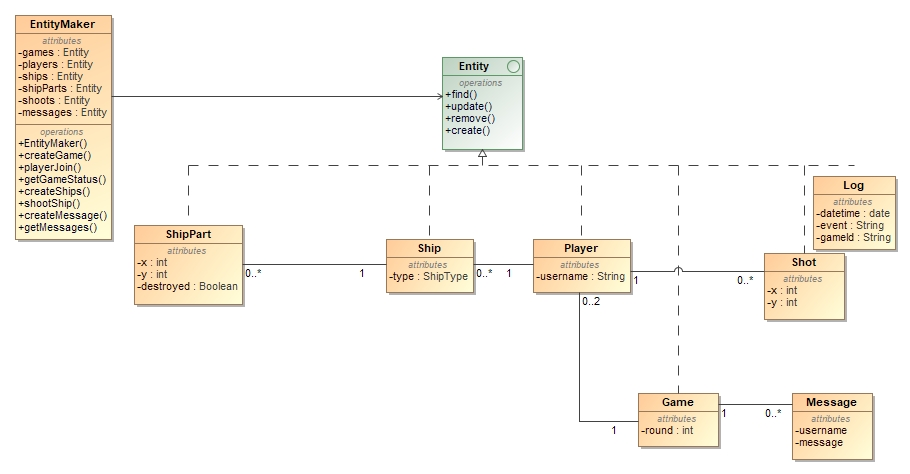
**public class** ConsoleLogSaverAdapter **implements** LoggerInterface {  
 @Override  
 **public void** save(JSONObject log) {  
 ConsoleLoggerService consoleLoggerService = **new** ConsoleLoggerService();  
  
 String logString = log.toString();  
  
 consoleLoggerService.save(logString);  
 }  
}

**public class** ConsoleLoggerService {  
 **public** ConsoleLoggerService() {  
 }  
  
 **public void** save(String text)  
 {  
 System.***out***.println(text);  
 }  
}

**public class** MysqlLoggerService {  
 **public** MysqlLoggerService() {  
 }  
  
 **public void** save(String gameId, String event)  
 {  
 Log log = **new** Log();  
 log.saveLog(gameId, event);  
 }  
}

## Facade

Pav. 15 facade šablono diagramos fragmentas



Šioje klasėje yra vienas „langelis“ tai yra EntityMaker klasė, per kurią yra komunikuojama su visais servisais. EntityMaker yra naudojamas daugiau nei 2-ose servisuose ir sub-system klasės irgi viršija skaičių 3.

**public interface** Entity {  
 JSONObject find(JSONObject json);  
 **void** update(JSONObject json);  
 **void** remove(JSONObject json);  
 JSONObject create(JSONObject json);  
}

**public class** EntityMaker {  
 **private** Entity **games**;  
 **private** Entity **players**;  
 **private** Entity **ships**;  
 **private** Entity **shipParts**;  
 **private** Entity **shoots**;  
 **private** Entity **messages**;  
 **private** Entity **logs**;  
  
 **private** MysqlAccess **mysqlAccess** = **null**;  
  
 **public** EntityMaker()  
 {  
 **this**.**games** = **new** Game();  
 **this**.**players** = **new** Player();  
 **this**.**ships** = **new** Ship();  
 **this**.**shipParts** = **new** ShipPart();  
 **this**.**shoots** = **new** Shoot();  
 **this**.**messages** = **new** Message();  
 **this**.**logs** = **new** Log();  
  
 **this**.**mysqlAccess** = MysqlAccess.*getInstance*();  
 }  
  
 **public** JSONObject createGame(JSONObject jsonInput)  
 {  
 JSONObject returnJson = **new** JSONObject();  
 returnJson = **this**.**games**.create(jsonInput);  
 **if**(returnJson.getInt(**"GameId"**) == -1)  
 {  
 returnJson = **new** JSONObject();  
 **return** returnJson.put(**"Error"**, **"Cannot create game"**);  
 }  
  
 **return** returnJson;  
 }  
  
 **public** JSONObject playerJoin(JSONObject jsonInput)  
 {  
 JSONObject returnJson = **this**.**players**.create(jsonInput);  
  
 **if** (returnJson.getInt(**"PlayerId"**) == -1) {  
 returnJson = **new** JSONObject();  
 returnJson.put(**"Error1"**, **"Maybe 2 players already joined the game?"**);  
 **return** returnJson;  
  
 }  
 **if** (returnJson.getInt(**"PlayerId"**) == -2) {  
 returnJson = **new** JSONObject();  
 returnJson.put(**"Error2"**, **"Username is already taken"**);  
 **return** returnJson;  
  
 }  
  
 **return** returnJson;  
 }  
**public** JSONObject getGameStatus(JSONObject jsonInput, Game game)  
 {  
 **return** game.getStatusWithRoundNumber(jsonInput);  
 }**public** JSONObject createShips(JSONObject jsonInput)  
 {  
 JSONObject returnJson = **new** JSONObject();  
 returnJson = **this**.**ships**.create(jsonInput);  
  
 **if**(returnJson.getInt(**"Status"**) == -1)  
 {  
 **return** returnJson.put(**"Error"**, **"Ships already placed"**);  
 }  
 **else if**(returnJson.getInt(**"Status"**) == -2)  
 {  
 **return** returnJson.put(**"Error"**, **"Placed ship count is too big or too small"**);  
 }  
 returnJson.remove(**"Status"**);  
  
 **return** returnJson;  
 }  
  
 **public** JSONObject shootShip(JSONObject jsonInput)  
 {  
 **return this**.**shoots**.create(jsonInput);  
 }

}

**public class** Game **implements** Entity {  
 @Override  
 **public** JSONObject find(JSONObject json) {  
 **return null**;  
 }  
  
 @Override  
 **public void** update(JSONObject json) {  
  
 }  
  
 @Override  
 **public void** remove(JSONObject json) {  
  
 }  
  
 @Override  
 **public** JSONObject create(JSONObject json) {  
 MysqlAccess mysqlAccess = MysqlAccess.*getInstance*();  
 JSONObject returnJson = **new** JSONObject();  
 String level = json.getString(**"Level"**);  
 **if**(!(level.equals(GameLevel.*EASY*) || level.equals(GameLevel.*MEDIUM*) || level.equals(GameLevel.*HARD*)))  
 {  
 **return** returnJson.put(**"GameId"**, -1);  
 }  
 **try** {  
 Statement statement = mysqlAccess.**connection**.createStatement();  
  
 statement.executeUpdate(**"INSERT INTO games (round, level) VALUES (0, \'"**+ level +**"\')"**);  
  
 ResultSet resultSet = statement.executeQuery(**"SELECT id FROM project.games ORDER BY id DESC LIMIT 1"**);  
 resultSet.next();  
 returnJson.put(**"GameId"**, resultSet.getInt(**"id"**));  
  
 LoggerInterface logger = **new** MysqlLogSaverAdapter();  
 logger.save(returnJson);  
  
 **return** returnJson; *// Current game id* } **catch** (SQLException throwables) {  
 throwables.printStackTrace();  
 }  
  
 **return** returnJson.put(**"GameId"**, -1);  
 }  
  
**public** JSONObject getStatusWithRoundNumber(JSONObject json)  
 {  
 MysqlAccess mysqlAccess = MysqlAccess.*getInstance*();  
 String gameId = (String) json.get(**"GameId"**);  
  
 JSONObject jsonObject = **new** JSONObject();  
  
 **try** {  
 Statement statement = mysqlAccess.**connection**.createStatement();  
  
 ResultSet resultSet = statement.executeQuery(**"SELECT** *COUNT***(***\****) as cnt FROM players WHERE fk\_game\_id="** + gameId);  
 resultSet.next();  
  
 **if** (resultSet.getInt(**"cnt"**) < 2) { *// Player count that joined to the game* jsonObject.append(**"GameStatus"**, **"Waiting players"**);  
 jsonObject.append(**"RoundNumber"**, 0);  
 **return** jsonObject;  
 }  
  
 resultSet = statement.executeQuery(**"SELECT** *COUNT***(DISTINCT fk\_player\_id) as playerCnt FROM ships "** +  
 **"INNER JOIN players ON players.id = ships.fk\_player\_id "** +  
 **"WHERE fk\_game\_id = "** + gameId);  
 resultSet.next();  
 **if** (resultSet.getInt(**"playerCnt"**) < 2) {  
 jsonObject.append(**"GameStatus"**, **"Waiting ships"**);  
 jsonObject.append(**"RoundNumber"**, 0);  
 **return** jsonObject;  
 }  
  
 resultSet = statement.executeQuery(**"SELECT** *\** **FROM games WHERE id="** + gameId);  
 resultSet.next();  
 **int** currentRound = resultSet.getInt(**"round"**);  
  
 resultSet = statement.executeQuery(**"SELECT DISTINCT fk\_player\_id as playerId, username FROM ships "** +  
 **"INNER JOIN ship\_parts ON ship\_parts.fk\_ship = ships.id "** +  
 **"INNER JOIN players ON players.id = fk\_player\_id "** +  
 **"WHERE destroyed = 0 AND fk\_game\_id = "** + gameId);  
 resultSet.next();  
  
 **if** (resultSet.isFirst() && !resultSet.isLast()) {  
 jsonObject.append(**"GameStatus"**, **"Live"**);  
 jsonObject.append(**"RoundNumber"**, currentRound);  
 **return** jsonObject;  
 } **else** {  
 jsonObject.append(**"GameStatus"**, **"Winner"**);  
 jsonObject.append(**"Username"**, resultSet.getString(**"username"**));  
 jsonObject.append(**"RoundNumber"**, currentRound);  
 **return** jsonObject;  
 }  
  
 } **catch** (SQLException throwables) {  
 throwables.printStackTrace();  
 }  
  
 **return null**;  
 }  
}

**public class** Log **implements** Entity {  
 @Override  
 **public** JSONObject find(JSONObject json) {  
 **return null**;  
 }  
  
 @Override  
 **public void** update(JSONObject json) {  
  
 }  
  
 @Override  
 **public void** remove(JSONObject json) {  
  
 }  
  
 @Override  
 **public** JSONObject create(JSONObject json) {  
 **return null**;  
 }  
  
 **public void** saveLog(String gameId, String event) {  
 MysqlAccess mysqlAccess = MysqlAccess.*getInstance*();  
 Statement statement = **null**;  
 **try** {  
 statement = mysqlAccess.**connection**.createStatement();  
 statement.executeUpdate(**"INSERT INTO logs (game\_id, event) VALUES (\'"**+ gameId +**"\', \'"** + event +**"\')"**);  
 } **catch** (SQLException throwables) {  
 throwables.printStackTrace();  
 }  
 }  
}

**public class** Message **implements** Entity {  
 @Override  
 **public** JSONObject find(JSONObject json) {  
 MysqlAccess mysqlAccess = MysqlAccess.*getInstance*();  
 JSONObject returnJson = **new** JSONObject();  
 String gameId = json.getString(**"GameId"**);  
 **try** {  
 Statement statement = mysqlAccess.**connection**.createStatement();  
  
 ResultSet resultSet = statement.executeQuery(**"SELECT nickname, message FROM messages WHERE fk\_game = "** + gameId);  
  
 JSONArray messages = **new** JSONArray();  
  
 **while** (resultSet.next()) {  
 String nickname = resultSet.getString(**"nickname"**);  
 String message = resultSet.getString(**"message"**);  
  
 JSONObject messageJson = **new** JSONObject();  
 messageJson.put(**"nickname"**, nickname);  
 messageJson.put(**"message"**, message);  
 messages.put(messageJson);  
 }  
  
 **return** returnJson.put(**"messages"**, messages);  
 } **catch** (SQLException throwables) {  
 throwables.printStackTrace();  
 }  
  
 **return** returnJson.put(**"error"**, **"error"**);  
 }  
  
 @Override  
 **public void** update(JSONObject json) {  
  
 }  
  
 @Override  
 **public void** remove(JSONObject json) {  
  
 }  
  
 @Override  
 **public** JSONObject create(JSONObject json) {  
 MysqlAccess mysqlAccess = MysqlAccess.*getInstance*();  
 JSONObject returnJson = **new** JSONObject();  
 **int** gameId = json.getInt(**"GameId"**);  
 String nick = json.getString(**"Nick"**);  
 String message = json.getString(**"Message"**);  
 **try** {  
 Statement statement = mysqlAccess.**connection**.createStatement();  
  
 statement.executeUpdate(**"INSERT INTO messages (nickname, message, fk\_game) VALUES (\'"** + nick + **"\', \'"**+ message +**"\', \'"** + gameId + **"\')"**);  
  
 returnJson.put(**"success"**, **"success"**);  
 **return** returnJson; *// Current game id* } **catch** (SQLException throwables) {  
 throwables.printStackTrace();  
 }  
  
 **return** returnJson.put(**"error"**, -1);  
 }  
}

**public class** Player **implements** Entity {  
 @Override  
 **public** JSONObject find(JSONObject json) {  
 **return null**;  
 }  
  
 @Override  
 **public void** update(JSONObject json) {  
  
 }  
  
 @Override  
 **public void** remove(JSONObject json) {  
  
 }  
  
 @Override  
 **public** JSONObject create(JSONObject json) {  
 MysqlAccess mysqlAccess = MysqlAccess.*getInstance*();  
 String username = (String) json.get(**"Username"**);  
 String gameId = (String) json.get(**"GameId"**);  
 JSONObject returnResult = **new** JSONObject();  
 **boolean** playerIsConnected = **false**;  
 **try** {  
 Statement statement = mysqlAccess.**connection**.createStatement();  
  
 ResultSet resultSet = statement.executeQuery(**"SELECT** *COUNT***(***\****) as cnt FROM players WHERE fk\_game\_id = "** + gameId);  
 resultSet.next();  
  
 **if** (resultSet.getInt(**"cnt"**) >= 2) {  
 **return** returnResult.put(**"PlayerId"**, -1);  
 }  
  
 **if**(resultSet.getInt(**"cnt"**) >= 1)  
 {  
 playerIsConnected = **true**;  
 }  
  
 resultSet = statement.executeQuery(**"SELECT** *COUNT***(***\****) as cnt FROM players WHERE fk\_game\_id = \'"** + gameId + **"\' AND username = \'"** + username + **"\' "**);  
 resultSet.next();  
  
 **if** (resultSet.getInt(**"cnt"**) >= 1) {  
 **return** returnResult.put(**"PlayerId"**, -2);  
 }  
  
  
  
 **if**(playerIsConnected) {  
 statement.executeUpdate(**"INSERT INTO players (username, fk\_game\_id) VALUES ( \'"** + username + **"\', \'"** + gameId + **"\')"**);  
  
 resultSet = statement.executeQuery(**"SELECT** *\** **FROM players WHERE fk\_game\_id = "** + gameId + **" ORDER BY id DESC LIMIT 1"**);  
 resultSet.next();  
 returnResult.put(**"PlayerId"**, resultSet.getInt(**"id"**));  
 resultSet = statement.executeQuery(**"SELECT** *\** **FROM games WHERE id = "** + gameId);  
 resultSet.next();  
 returnResult.put(**"Level"**, resultSet.getString(**"level"**));  
 **return** returnResult; *// new player unique id* }  
 **else**{  
 resultSet = statement.executeQuery(**"SELECT** *\** **FROM players ORDER BY id DESC LIMIT 1"**);  
  
 **if**(!resultSet.next()) {  
 statement.executeUpdate(**"INSERT INTO players (username, fk\_game\_id) VALUES (\'"** + username + **"\', \'"** + gameId + **"\')"**);  
 }  
 **else**{  
  
 Integer index = resultSet.getInt(**"id"**);  
 Random rand = **new** Random();  
 **int** randomNum = rand.nextInt((2 - 1) + 1) + 1;  
 index += randomNum;  
 statement.executeUpdate(**"INSERT INTO players (id, username, fk\_game\_id) VALUES (\'"**+ index +**"\', \'"** + username + **"\', \'"** + gameId + **"\')"**);  
  
 }  
  
 resultSet = statement.executeQuery(**"SELECT** *\** **FROM players WHERE fk\_game\_id = "** + gameId + **" ORDER BY id DESC LIMIT 1"**);  
 resultSet.next();  
 returnResult.put(**"PlayerId"**, resultSet.getInt(**"id"**));  
 resultSet = statement.executeQuery(**"SELECT** *\** **FROM games WHERE id = "** + gameId);  
 resultSet.next();  
 returnResult.put(**"Level"**, resultSet.getString(**"level"**));  
 **return** returnResult; *// new player unique id* }  
  
  
 } **catch** (SQLException throwables) {  
 throwables.printStackTrace();  
 }  
  
 **return** returnResult.put(**"PlayerId"**, -1);  
 }  
}

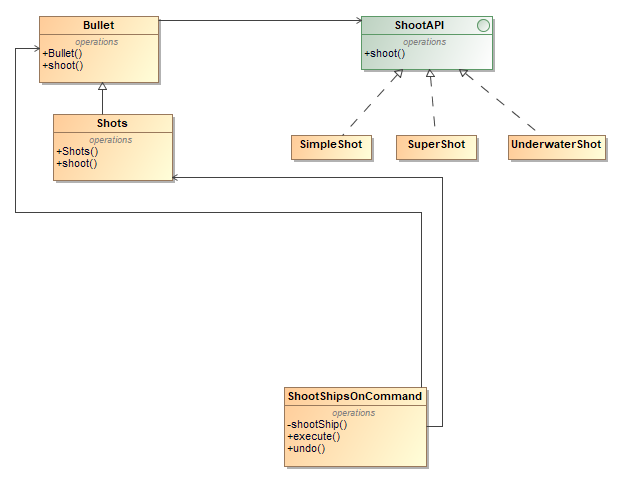
**public class** Ship **implements** Entity {  
 @Override  
 **public** JSONObject find(JSONObject json) {  
 **return null**;  
 }  
  
 @Override  
 **public void** update(JSONObject json) {  
  
 }  
  
 @Override  
 **public void** remove(JSONObject json) {  
  
 }  
  
 @Override  
 **public** JSONObject create(JSONObject json) {  
 MysqlAccess mysqlAccess = MysqlAccess.*getInstance*();  
 String playerId = (String) json.get(**"PlayerId"**);  
 String gameId = (String) json.get(**"GameId"**);  
  
 JSONObject jsonShips = (JSONObject) json.get(**"Ships"**);  
 JSONArray keys = jsonShips.names();  
 JSONObject returnObject = **new** JSONObject();  
 **try** {  
 Statement statement = mysqlAccess.**connection**.createStatement();  
  
 ResultSet set = statement.executeQuery(**"SELECT** *COUNT***(***\****) as cnt FROM ships WHERE fk\_player\_id = "** + playerId);  
 set.next();  
  
 **if** (set.getInt(**"cnt"**) >= 1) {  
 **return** returnObject.put(**"Status"**, -1);  
  
 }  
  
 set = statement.executeQuery(**"SELECT level FROM games WHERE id = "** + gameId);  
 set.next();  
 String level = set.getString(**"Level"**);  
 **if**(keys.length() != GameLevel.*SHIPEASYCOUNT* && level.equals(GameLevel.*EASY*)) {  
 **return** returnObject.put(**"Status"**, -2);  
 }  
  
 **if**(keys.length() != GameLevel.*SHIPMEDIUMCOUNT* && level.equals(GameLevel.*MEDIUM*)) {  
 **return** returnObject.put(**"Status"**, -2);  
 }  
  
 **if**(keys.length() != GameLevel.*SHIPHARDCOUNT* && level.equals(GameLevel.*HARD*)){  
 **return** returnObject.put(**"Status"**, -2);  
 }  
  
 **for** (**int** i = 0; i < keys.length(); i++) {  
  
 String key = keys.get(i).toString();  
 JSONObject ship = (JSONObject) jsonShips.get(key);  
  
 String shipType = (String) ship.get(**"type"**);  
 Boolean armour = (Boolean) ship.get(**"armour"**);  
 statement.executeUpdate(**"INSERT INTO ships (fk\_player\_id, type, radar\_off) VALUES (\'"** + playerId + **"\', \'"**+ shipType +**"\', '0' )"**);  
  
 ResultSet resultSet = statement.executeQuery(**"SELECT** *\** **FROM ships WHERE fk\_player\_id = "** + playerId + **" ORDER BY id DESC LIMIT 1"**);  
 resultSet.next();  
  
 **int** shipId = resultSet.getInt(**"id"**);  
 JSONArray coordinates = (JSONArray) ship.get(**"coord"**);  
 **for**(**int** coordCnt = 0; coordCnt < coordinates.length(); coordCnt++) {  
 String[] splitCoords = coordinates.get(coordCnt).toString().split(**","**);  
 **if**(armour) {  
 statement.executeUpdate(**"INSERT INTO ship\_parts (x, y, armour, destroyed, fk\_ship) VALUES (\'"** + splitCoords[0] + **"\', \'"** + splitCoords[1] + **"\', '1', '0', \'"** + shipId + **"\') "**);  
 }  
 **else** {  
 statement.executeUpdate(**"INSERT INTO ship\_parts (x, y, armour, destroyed, fk\_ship) VALUES (\'"** + splitCoords[0] + **"\', \'"** + splitCoords[1] + **"\', '0', '0', \'"** + shipId + **"\') "**);  
 }  
 }  
 }  
 }**catch** (SQLException throwables) {  
 throwables.printStackTrace();  
 }  
  
 **return** returnObject.put(**"Status"**, 1);  
 }  
}

**public class** ShipPart **implements** Entity {  
 @Override  
 **public** JSONObject find(JSONObject json) {  
 **return null**;  
 }  
  
 @Override  
 **public void** update(JSONObject json) {  
  
 }  
  
 @Override  
 **public void** remove(JSONObject json) {  
  
 }  
  
 @Override  
 **public** JSONObject create(JSONObject json) {  
 **return new** JSONObject();  
 }  
}

**public class** Shoot **implements** Entity {  
 @Override  
 **public** JSONObject find(JSONObject json) {  
 **return null**;  
 }  
  
 @Override  
 **public void** update(JSONObject json) {  
  
 }  
  
 @Override  
 **public void** remove(JSONObject json) {  
  
 }  
  
 @Override  
 **public** JSONObject create(JSONObject json) {  
 MysqlAccess mysqlAccess = MysqlAccess.*getInstance*();  
 JSONObject result = **new** JSONObject();  
 **boolean** error = **false**;  
 String playerId = (String) json.get(**"PlayerId"**);  
 String gameId = (String) json.get(**"GameId"**);  
 JSONObject shoot = (JSONObject) json.get(**"Shoot"**);  
 JSONArray coordinates = (JSONArray) shoot.getJSONArray(**"Coord"**);  
 String type = (String) shoot.get(**"Type"**);  
 **try** {  
 Statement statement = mysqlAccess.**connection**.createStatement();  
  
  
 ResultSet resultSet = statement.executeQuery(**"SELECT id FROM players WHERE fk\_game\_id = "**+ gameId +**" AND id != "** + playerId);  
 resultSet.next();  
 **int** enemyId = resultSet.getInt(**"id"**);  
 JSONArray hitCoordinates = **new** JSONArray();  
 JSONArray armourCoordinates = **new** JSONArray();  
 JSONArray missCoordinates = **new** JSONArray();  
 **boolean** isShipDestroyed = **false**;  
 **for** (**int** i = 0; i < coordinates.length(); i++) {  
 String[] splitCoords = coordinates.get(i).toString().split(**","**);  
*// error = false;  
// resultSet = statement.executeQuery("SELECT COUNT(id) as cnt FROM shoots WHERE fk\_player\_id = "+ playerId +" AND x = "+ splitCoords[0] +" AND y = " + splitCoords[1]);  
// resultSet.next();  
// Integer shootCordinateCheck = resultSet.getInt("cnt");  
// if(!error && shootCordinateCheck > 0) {  
// error = true;  
// }  
  
// System.out.println(coordinates.getString(i));* **if**(!error) {  
 resultSet = statement.executeQuery(**"SELECT ship\_parts.id as part\_id, ships.id as ship\_id, ship\_parts.armour as armour, ship\_parts.x as xCord, ship\_parts.y as yCord FROM ships INNER JOIN ship\_parts ON ship\_parts.fk\_ship = ships.id WHERE ship\_parts.destroyed != 1 AND ship\_parts.x = "** + splitCoords[0] + **" AND y = "** + splitCoords[1] + **" AND ships.fk\_player\_id = "** + enemyId + **" AND ships.radar\_off = 0 AND ships.type != 'SUBMARINE' "**);  
*//* **if** (resultSet.next() != **false**) {  
 String partId = resultSet.getString(**"part\_id"**);  
 String shipId = resultSet.getString(**"ship\_id"**);  
 Integer x = resultSet.getInt(**"xCord"**);  
 Integer y = resultSet.getInt(**"yCord"**);  
 String shootCoordinates = String.*format*(**"%d,%d"**, x, y);  
 Integer armour = resultSet.getInt(**"armour"**);  
 **if**(armour > 0) {  
  
 armour--;  
 statement.executeUpdate(**"UPDATE ship\_parts SET armour = \'"** + armour + **"\' WHERE id = "** + partId);  
 armourCoordinates.put(shootCoordinates);  
  
 }  
 **else** {  
 statement.executeUpdate(**"UPDATE ship\_parts SET destroyed = '1' WHERE id = "** + partId);  
 hitCoordinates.put(shootCoordinates);  
  
 resultSet = statement.executeQuery(**"SELECT** *COUNT***(id) as cnt FROM ship\_parts WHERE fk\_ship = "** + shipId + **" AND destroyed = 1"**);  
 resultSet.next();  
 Integer destroyShipCount = resultSet.getInt(**"cnt"**);  
  
 resultSet = statement.executeQuery(**"SELECT** *COUNT***(id) as cnt FROM ship\_parts WHERE fk\_ship = "** + shipId);  
 resultSet.next();  
 Integer allShipCount = resultSet.getInt(**"cnt"**);  
  
 **if** (destroyShipCount == allShipCount && !isShipDestroyed) {  
 isShipDestroyed = **true**;  
 }  
 }  
*//* }  
 **else** {  
 **if**(type.equals(**"Underwater"**) || type.equals(**"Super"**))  
 {  
 resultSet = statement.executeQuery(**"SELECT ship\_parts.id as part\_id, ships.id as ship\_id, ship\_parts.armour as armour, ship\_parts.x as xCord, ship\_parts.y as yCord FROM ships INNER JOIN ship\_parts ON ship\_parts.fk\_ship = ships.id WHERE ship\_parts.destroyed != 1 AND ship\_parts.x = "** + splitCoords[0] + **" AND y = "** + splitCoords[1] + **" AND ships.fk\_player\_id = "** + enemyId + **" AND ships.radar\_off = 0 AND ships.type = 'SUBMARINE' "**);  
 **if** (resultSet.next() != **false**) {  
 String partId = resultSet.getString(**"part\_id"**);  
 String shipId = resultSet.getString(**"ship\_id"**);  
 Integer x = resultSet.getInt(**"xCord"**);  
 Integer y = resultSet.getInt(**"yCord"**);  
 String shootCoordinates = String.*format*(**"%d,%d"**, x, y);  
 Integer armour = resultSet.getInt(**"armour"**);  
 **if** (armour > 0) {  
  
 armour--;  
 statement.executeUpdate(**"UPDATE ship\_parts SET armour = \'"** + armour + **"\' WHERE id = "** + partId);  
 armourCoordinates.put(shootCoordinates);  
  
 } **else** {  
 statement.executeUpdate(**"UPDATE ship\_parts SET destroyed = '1' WHERE id = "** + partId);  
 hitCoordinates.put(shootCoordinates);  
  
 resultSet = statement.executeQuery(**"SELECT** *COUNT***(id) as cnt FROM ship\_parts WHERE fk\_ship = "** + shipId + **" AND destroyed = 1"**);  
 resultSet.next();  
 Integer destroyShipCount = resultSet.getInt(**"cnt"**);  
  
 resultSet = statement.executeQuery(**"SELECT** *COUNT***(id) as cnt FROM ship\_parts WHERE fk\_ship = "** + shipId);  
 resultSet.next();  
 Integer allShipCount = resultSet.getInt(**"cnt"**);  
  
 **if** (destroyShipCount == allShipCount && !isShipDestroyed) {  
 isShipDestroyed = **true**;  
 }  
 }  
 }  
 **else**{  
 Integer x = Integer.*parseInt*(splitCoords[0]);  
 Integer y = Integer.*parseInt*(splitCoords[1]);  
 String shootCoordinates = String.*format*(**"%d,%d"**, x, y);  
 missCoordinates.put(shootCoordinates);  
 }  
 }  
 **else** {  
 Integer x = Integer.*parseInt*(splitCoords[0]);  
 Integer y = Integer.*parseInt*(splitCoords[1]);  
 String shootCoordinates = String.*format*(**"%d,%d"**, x, y);  
 missCoordinates.put(shootCoordinates);  
 }  
 }  
statement.executeUpdate(**"INSERT INTO shoots (x, y, type, fk\_player\_id) VALUES (\'"** + splitCoords[0] + **"\', \'"** + splitCoords[1] + **"\', \'"**+ type +**"\', \'"** + playerId + **"\')"**);  
 }  
 }  
 **if** (hitCoordinates.length() > 0) {  
 result.put(**"Hit"**, hitCoordinates);  
 }  
 **if** (armourCoordinates.length() > 0) {  
 result.put(**"Armour"**, armourCoordinates);  
 }  
  
 **if**(missCoordinates.length() > 0)  
 {  
 result.put(**"Miss"**, missCoordinates);  
 }  
  
 **if**(missCoordinates.length() > 0)  
 {  
 resultSet = statement.executeQuery(**"SELECT round FROM games WHERE id = "** + gameId);  
 resultSet.next();  
 **int** roundCount = Integer.*parseInt*(resultSet.getString(**"round"**));  
 roundCount++;  
 statement.executeUpdate(**"UPDATE games SET round = \'"**+ roundCount +**"\' WHERE id = "** + gameId);  
 }  
 result.put(**"IsShipDestroyed"**, isShipDestroyed);  
  
 }**catch** (SQLException throwables) {  
 throwables.printStackTrace();  
 }  
  
 **return** result;  
 }  
  
}

## Bridge

Pav. 16 bridge šablono diagramos fragmentas



Nuotraukoje matome kaip naudojamas tilto (angl. bridge) šablonas skirtingu šūvių kūrimui. Yra sukurtos dvi abstrakcijos klasės ir keturios realizacijos klasės.

**package** client.Models.Bridge;  
  
**import** org.json.JSONObject;  
  
**public abstract class** Bullet {  
 **protected** ShootAPI **shootAPI**;  
  
 **protected** Bullet(ShootAPI shootAPI){  
 **this**.**shootAPI** = shootAPI;  
 }  
 **public abstract** JSONObject shoot();  
}

**package** client.Models.Bridge;  
  
**import** client.Models.ConnectionManager;  
**import** org.json.JSONObject;  
  
**public interface** ShootAPI {  
 **public** JSONObject shoot(String coordinates, Integer gameId, Integer playerId, ConnectionManager connectionManager);  
}

**package** client.Models.Bridge;  
  
**import** client.Models.ConnectionManager;  
**import** org.json.JSONObject;  
  
**public class** Shots **extends** Bullet {  
 **private** String **coordinates**;  
 **private int gameId**, **playerId**;  
 **private** ConnectionManager **connectionManager**;  
  
 **public** Shots(String coordinates, Integer gameId, Integer playerId, ConnectionManager connectionManager, ShootAPI shootAPI ){  
 **super**(shootAPI);  
 **this**.**coordinates** = coordinates;  
 **this**.**gameId** = gameId;  
 **this**.**playerId** = playerId;  
 **this**.**connectionManager** = connectionManager;  
 }  
  
 **public** JSONObject shoot() {  
 **return shootAPI**.shoot(**coordinates**, **gameId**, **playerId**, **connectionManager**);  
 }  
  
}

**package** client.Models.Bridge;  
  
**import** client.Models.ConnectionManager;  
**import** org.json.JSONArray;  
**import** org.json.JSONObject;  
  
**public class** SimpleShot **implements** ShootAPI {  
  
 **public** JSONObject shoot(String coordinates, Integer gameId, Integer playerId, ConnectionManager connectionManager){  
 JSONObject object = **new** JSONObject();  
 JSONObject shootListObject = **new** JSONObject();  
 JSONArray shootCoordinatesArray = **new** JSONArray();  
 shootCoordinatesArray.put(coordinates);  
 shootListObject.put(**"Coord"**, shootCoordinatesArray);  
 shootListObject.put(**"Type"**, **"Simple"**);  
  
 object.put(**"Shoot"**, shootListObject);  
 object.put(**"GameId"**, gameId.toString());  
 object.put(**"PlayerId"**, playerId.toString());  
*// System.out.println(object.toString());* JSONObject data = connectionManager.getRequest(**"/api/ships/shoot"**, object.toString());  
 **return** data;  
 }  
  
  
}

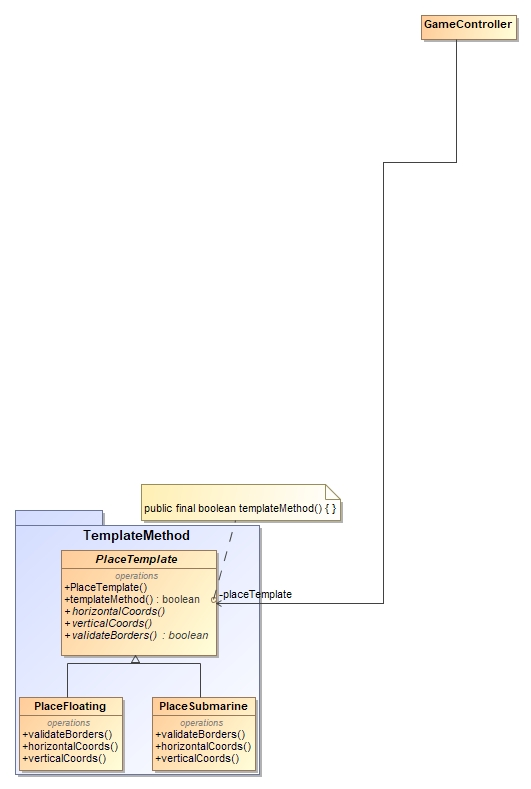
**package** client.Models.Bridge;  
  
**import** client.Models.ConnectionManager;  
**import** client.Models.GameData;  
**import** org.json.JSONArray;  
**import** org.json.JSONObject;  
  
**public class** SuperShot **implements** ShootAPI {  
 @Override  
 **public** JSONObject shoot(String coordinates, Integer gameId, Integer playerId, ConnectionManager connectionManager){  
 GameData.*RoundShootSuper* = GameData.*RoundId* + 8;  
 Integer x = 0;  
 Integer y = 0;  
 String[] splitCoords = coordinates.split(**","**);  
 x = Integer.*parseInt*(splitCoords[0]);  
 y = Integer.*parseInt*(splitCoords[1]);  
  
 JSONObject object = **new** JSONObject();  
 JSONObject shootListObject = **new** JSONObject();  
 JSONArray shootCoordinatesArray = **new** JSONArray();  
  
 shootCoordinatesArray.put(coordinates);  
 *// sovimo coordinates*  
**if**(x+1 < GameData.*gameLevel*.getGridLength()) {  
  
 String shotCoordinates = **""** + (x + 1) + **","** + y;  
 shootCoordinatesArray.put(shotCoordinates);  
 }  
 **if**(y + 1 < GameData.*gameLevel*.getGridHeight()) {  
 String shotCoordinates = **""** + x + **","** + (y + 1);  
 shootCoordinatesArray.put(shotCoordinates);  
 }  
  
 **if**(x+1 < GameData.*gameLevel*.getGridLength() && y+1 < GameData.*gameLevel*.getGridHeight()) {  
 String shotCoordinates = **""** + (x + 1) + **","** + (y+1);  
 shootCoordinatesArray.put(shotCoordinates);  
 }  
  
 *//*  
shootListObject.put(**"Coord"**, shootCoordinatesArray);  
 shootListObject.put(**"Type"**, **"Super"**);  
  
 object.put(**"Shoot"**, shootListObject);  
 object.put(**"GameId"**, gameId.toString());  
 object.put(**"PlayerId"**, playerId.toString());  
*// System.out.println(object.toString());*  
JSONObject data = connectionManager.getRequest(**"/api/ships/shoot"**, object.toString());  
 **return** data;  
 }  
}

**package** client.Models.Bridge;  
  
**import** client.Models.ConnectionManager;  
**import** client.Models.GameData;  
**import** org.json.JSONArray;  
**import** org.json.JSONObject;  
  
**public class** UnderwaterShot **implements** ShootAPI {  
 @Override  
 **public** JSONObject shoot(String coordinates, Integer gameId, Integer playerId, ConnectionManager connectionManager){  
 JSONObject object = **new** JSONObject();  
 JSONObject shootListObject = **new** JSONObject();  
 JSONArray shootCoordinatesArray = **new** JSONArray();  
 shootCoordinatesArray.put(coordinates);  
 shootListObject.put(**"Coord"**, shootCoordinatesArray);  
 shootListObject.put(**"Type"**, **"Underwater"**);  
  
 object.put(**"Shoot"**, shootListObject);  
 object.put(**"GameId"**, gameId.toString());  
 object.put(**"PlayerId"**, playerId.toString());  
*// System.out.println(object.toString());* JSONObject data = connectionManager.getRequest(**"/api/ships/shoot"**, object.toString());  
 **return** data;  
 }  
}

# Antrojo etapo projektavimo šablonai

## Template method

pav. template method šablono diagramos fragmentas



Šį šabloną pritaikėme laivų padėjimo validacijai. Kadangi jei laivas plūduriuojantis, jis negali būti padėtas, jei aplinkui laivą nėra vieno langelio laisvos vietos, o povandeninis laivas gali būti padėtas po kitu laivu, todėl tas pats validacijos metodas jau netinka ir reikia laivo padėjimo validaciją atlikti kitaip. Taip pat templateMethod yra hook operacija, kuri priklausomai nuo patikros vykdo tam tikrą funkcionalumą, čia veiksmų sekos keisti negalima ir ji bus atliekama visada taip kaip yra aprašyta.

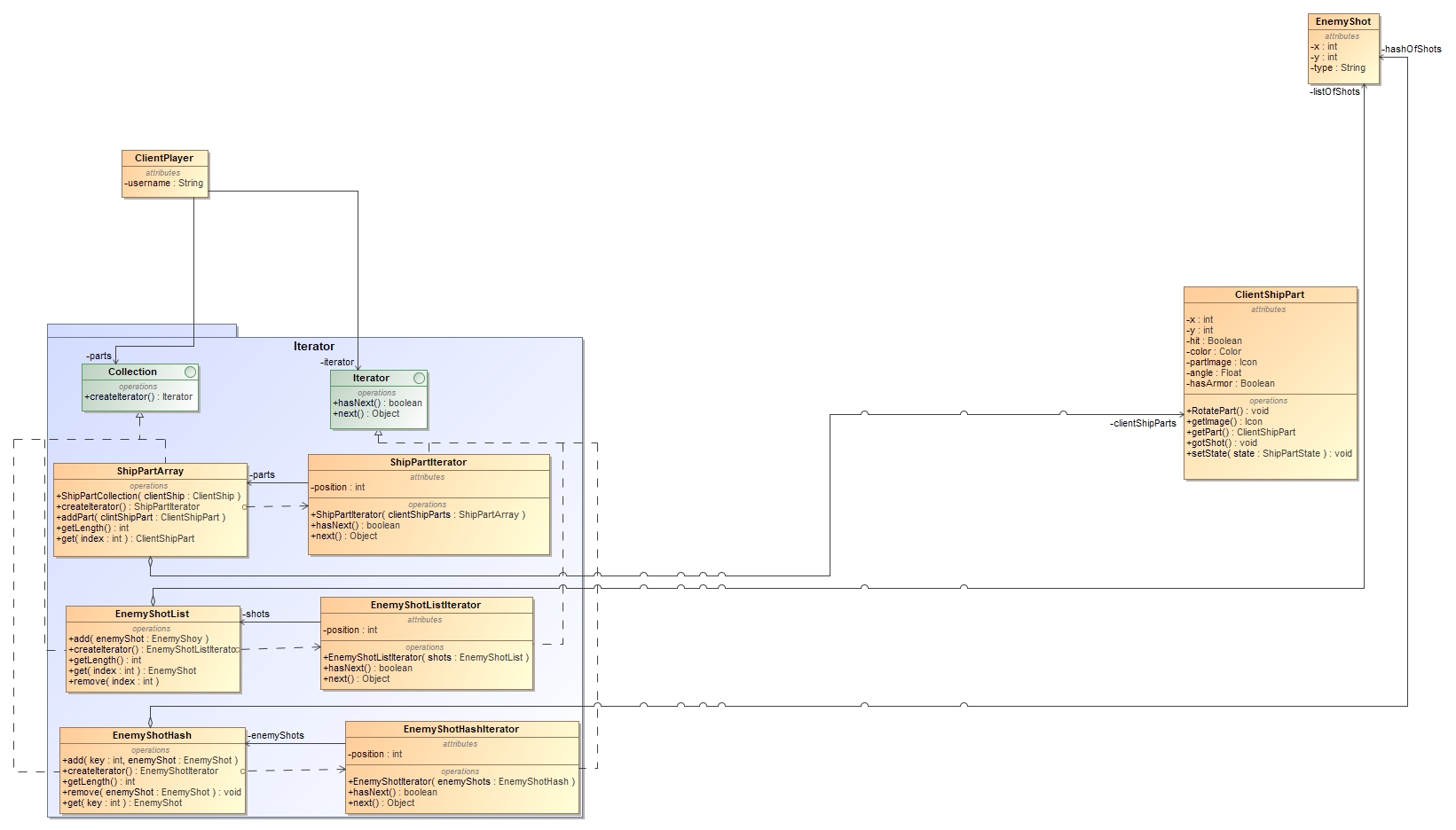
**package** client.Models.TemplateMethod;  
  
**import** client.GameConstants.GameConstants;  
**import** client.Models.GameData;  
**import** client.Views.SeaButton;  
  
**public abstract class** PlaceTemplate {  
 **public** PlaceTemplate() {  
  
 }  
  
 **public final boolean** templateMethod(SeaButton[][][] seaButtons, SeaButton seaButton, **int** gridCountRows, **int** gridCountCol, **int** callerIndex, **int** shipLength) {  
 **boolean** check = **true**;  
 *// Hook* **if** (GameData.*Rotation*.equals(GameConstants.***rotationHorizontal***)) {  
 **for**(**int** plus = 0; plus < shipLength; plus++) {  
 horizontalCoords(seaButton, plus);  
 check = validateBorders(seaButtons, gridCountRows, gridCountCol, callerIndex, check);  
 }  
 }  
 **else** {  
 **for**(**int** plus = 0; plus < shipLength; plus++) {  
 verticalCoords(seaButton, plus);  
 check = validateBorders(seaButtons, gridCountRows, gridCountCol, callerIndex, check);  
 }  
 }  
 **return** check;  
 }  
  
 *// Step1* **public abstract void** horizontalCoords(SeaButton seaButton, **int** plus);  
  
 *// Step2* **public abstract void** verticalCoords(SeaButton seaButton, **int** plus);  
  
 *// Abstract step3* **public abstract boolean** validateBorders(SeaButton[][][] seaButtons, **int** gridCountRows, **int** gridCountCol, **int** callerGridIndex, **boolean** check);  
}

**package** client.Models.TemplateMethod;  
  
**import** client.Views.SeaButton;  
  
**import** java.awt.\*;  
  
**public class** PlaceFloating **extends** PlaceTemplate {  
 **private int xOriginal**, **yOriginal**;  
  
 @Override  
 **public boolean** validateBorders(SeaButton[][][] seaButtons, **int** gridCountRows, **int** gridCountCol, **int** callerGridIndex, **boolean** check) {  
 **for** (**int** x = -1; x < 2; x++) {  
 **for** (**int** y = -1; y < 2; y++) {  
 **if** ((y != 0 && x == 0) || (x != 0 && y == 0) || (x != 0 && y != 0)) {  
 Integer xCord = **xOriginal** + x;  
 Integer yCord = **yOriginal** + y;  
  
 **if** (xCord > -1 && yCord > -1 && xCord < gridCountRows && yCord < gridCountCol) {  
 **if** (seaButtons[callerGridIndex][xCord][yCord].getColor() == Color.***yellow***) {  
 check = **false**;  
 }  
 **if**((**xOriginal** < 0 || **xOriginal** >= gridCountRows) || (**yOriginal** < 0 || **yOriginal** >= gridCountCol))  
 {  
 check = **false**;  
 }  
 }  
  
 }  
  
 }  
  
 }  
 **return** check;  
 }  
  
 @Override  
 **public void** horizontalCoords(SeaButton seaButton, **int** plus) {  
 **xOriginal** = seaButton.getGridX();  
 **yOriginal** = seaButton.getGridY() + plus;  
 }  
  
 @Override  
 **public void** verticalCoords(SeaButton seaButton, **int** plus) {  
 **xOriginal** = seaButton.getGridX() + plus;  
 **yOriginal** = seaButton.getGridY();  
 }  
}

**package** client.Models.TemplateMethod;  
  
**import** client.Views.SeaButton;  
  
**import** java.awt.\*;  
  
**public class** PlaceSubmarine **extends** PlaceTemplate {  
 **private int xOriginal**, **yOriginal**;  
  
 @Override  
 **public boolean** validateBorders(SeaButton[][][] seaButtons, **int** gridCountRows, **int** gridCountCol, **int** callerGridIndex, **boolean** check) {  
 **for** (**int** x = -1; x < 2; x++) {  
 **for** (**int** y = -1; y < 2; y++) {  
 **if** ((y != 0 && x == 0) || (x != 0 && y == 0) || (x != 0 && y != 0)) {  
 Integer xCord = **xOriginal** + x;  
 Integer yCord = **yOriginal** + y;  
  
 **if** (xCord > -1 && yCord > -1 && xCord < gridCountRows && yCord < gridCountCol) {  
 **if**((**xOriginal** < 0 || **xOriginal** >= gridCountRows) || (**yOriginal** < 0 || **yOriginal** >= gridCountCol))  
 {  
 check = **false**;  
 }  
 }  
  
 }  
  
 }  
  
 }  
 **return** check;  
 }  
  
 @Override  
 **public void** horizontalCoords(SeaButton seaButton, **int** plus) {  
 **xOriginal** = seaButton.getGridX();  
 **yOriginal** = seaButton.getGridY() + plus;  
 }  
  
 @Override  
 **public void** verticalCoords(SeaButton seaButton, **int** plus) {  
 **xOriginal** = seaButton.getGridX() + plus;  
 **yOriginal** = seaButton.getGridY();  
 }  
}

## Iterator

pav. iterator šablono diagramos fragmentas



Šį šabloną naudojame, kai norime pereiti per skirtingas duomenų kolekcijas tuo pačiu būdu, pvz. nuo pirmo iki paskutinio, todėl šis šablonas buvo pritaikytas pereiti per trijų skirtingų tipų duomenų struktūras. Šio šablono pagalba galime susikurti norimą kolekciją ir per ją nesunkiai pereiti naudodami for ciklą, bei patikrinimą „ar yra kitas elementas?“.

**package** client.Models.Iterator;  
  
**public interface** Collection {  
 Iterator createIterator();  
}

**package** client.Models.Iterator;  
  
**public interface** Iterator {  
 **boolean** hasNext();  
 Object next();  
}

**package** client.Models.Iterator;  
  
**import** client.Models.Observer.EnemyShot;  
  
**import** java.util.HashMap;  
**import** java.util.Map;  
  
**public class** EnemyShotHash **implements** Collection {  
 **private** HashMap<Integer, EnemyShot> **hashOfShots** = **new** HashMap<>();  
  
 **public** EnemyShotHash() {  
  
 }  
  
 **public** EnemyShotHash(HashMap<EnemyShot, Integer> hashOfShots) {  
 **for** (Map.Entry<EnemyShot, Integer> entry : hashOfShots.entrySet())  
 **this**.**hashOfShots**.put(entry.getValue(), entry.getKey());  
 }  
  
 **public void** add(**int** key, EnemyShot enemyShot) {  
 **hashOfShots**.put(key, enemyShot);  
 }  
  
 **public int** getLength() {  
 **return hashOfShots**.size();  
 }  
  
 **public** EnemyShot get(**int** index) {  
 **return hashOfShots**.get(index);  
 }  
  
 **public void** remove(EnemyShot enemyShot) {  
 **hashOfShots**.remove(enemyShot);  
 }  
  
 **public** HashMap<Integer, EnemyShot> getEnemyShots() {  
 **return hashOfShots**;  
 }  
  
 @Override  
 **public** EnemyShotHashIterator createIterator() {  
 **return new** EnemyShotHashIterator(**this**);  
 }  
}

**package** client.Models.Iterator;  
  
**public class** EnemyShotHashIterator **implements** Iterator {  
 EnemyShotHash **enemyShots**;  
 **int position** = 1;  
  
 **public** EnemyShotHashIterator(EnemyShotHash enemyShots) {  
 **this**.**enemyShots** = enemyShots;  
 }  
  
 @Override  
 **public boolean** hasNext()  
 {  
 **if** (**position** > **enemyShots**.getLength() || **enemyShots**.get(**position**) == **null**)  
 **return false**;  
 **else  
 return true**;  
 }  
  
 @Override  
 **public** Object next() {  
 **return enemyShots**.get(**position**++);  
 }  
}

**package** client.Models.Iterator;  
  
**import** client.Models.Observer.EnemyShot;  
  
**import** java.util.ArrayList;  
  
**public class** EnemyShotList **implements** Collection {  
 **private** ArrayList<EnemyShot> **listOfShots** = **new** ArrayList<>();  
  
 **public** EnemyShotList() {  
  
 }  
  
 **public** EnemyShotList(ArrayList<EnemyShot> listOfShots) {  
 **this**.**listOfShots** = listOfShots;  
 }  
  
 **public void** add(EnemyShot enemyShot) {  
 **listOfShots**.add(enemyShot);  
 }  
  
 **public int** getLength() {  
 **return listOfShots**.size();  
 }  
  
 **public** EnemyShot get(**int** index) {  
 **return** listOfShots.get(index);  
 }  
  
 **public void** remove(**int** index) {  
 listOfShots.remove(index);  
 }  
  
 **public** ArrayList<EnemyShot> getEnemyShots() {  
 **return** listOfShots;  
 }  
  
 @Override  
 **public** EnemyShotListIterator createIterator() {  
 **return new** EnemyShotListIterator(**this**);  
 }  
}

**package** client.Models.Iterator;  
  
**public class** EnemyShotListIterator **implements** Iterator {  
 EnemyShotList **shots**;  
 **int position** = 0;  
  
 **public** EnemyShotListIterator(EnemyShotList shots) {  
 **this**.**shots** = shots;  
 }  
  
 @Override  
 **public boolean** hasNext()  
 {  
 **if** (**position** >= **shots**.getLength() || **shots**.get(**position**) == **null**)  
 **return false**;  
 **else  
 return true**;  
 }  
  
 @Override  
 **public** Object next() {  
 **return shots**.get(**position**++);  
 }  
}

**package** client.Models.Iterator;  
  
**import** client.Models.ClientShip;  
**import** client.Models.ClientShipPart;  
  
**public class** ShipPartArray **implements** Collection {  
 **private** ClientShipPart[] **clientShipParts**;  
 **private int amountOfParts** = 0;  
 **private int size**;  
  
 **public** ShipPartArray(ClientShip clientShip) {  
 **size** = clientShip.getShipPartCount();  
 **clientShipParts** = clientShip.getPartsAsArray();  
 }  
  
 **public void** addPart(ClientShipPart clientShipPart) {  
 **if** (**amountOfParts** >= **size**)  
 System.***out***.println(**"ClientShipPart collection is full"**);  
 **else** {  
 **clientShipParts**[**amountOfParts**++] = clientShipPart;  
 }  
 }  
  
 **public int** getLength() {  
 **return clientShipParts**.**length**;  
 }  
  
 **public** ClientShipPart get(**int** index) {  
 **return clientShipParts**[index];  
 }  
  
 @Override  
 **public** ShipPartIterator createIterator() {  
 **return new** ShipPartIterator(**this**);  
 }  
}

**package** client.Models.Iterator;  
  
**public class** ShipPartIterator **implements** Iterator {  
 ShipPartArray **clientShipParts**;  
 **int position** = 0;  
  
 **public** ShipPartIterator(ShipPartArray clientShipParts) {  
 **this**.**clientShipParts** = clientShipParts;  
 }  
  
 @Override  
 **public boolean** hasNext()  
 {  
 **if** (**position** >= **clientShipParts**.getLength() || **clientShipParts**.get(**position**) == **null**)  
 **return false**;  
 **else  
 return true**;  
 }  
  
 @Override  
 **public** Object next() {  
 **return clientShipParts**.get(**position**++);  
 }  
}

## Flyweight

pav. flyweight šablono diagramos fragmentas

A picture containing Word

Description automatically generated

Flyweight šablono pagalba, gauname laivų nuotraukas. Šis šablonas padeda mums atsikratyti perteklinio nuotraukų nuskaitymo ir jų išsaugojimo atmintyje. Taip pat šis šablonas atlaisvina daugiau atminties ir tuo pačiu pagreitina nuotraukų atsiradimą ekrane.

Atlikus greitaveikos ir atminties naudojimo matavimus gavome tokius rezultatus.

pav. flyweight šablono greitaveikos ir atminties naudojimo matavimas

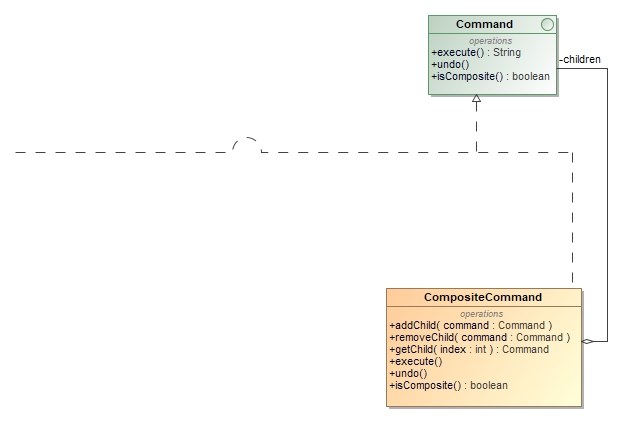
Graphical user interface

Description automatically generated

Kadangi, kolkas mūsų žaidimas neturi tiek daug laivų vienu metu ekrane, tai atminties ir greitaveikos nauda nėra tokia akivaizdi. Todėl greitaveikos ir atminties naudojimo matavimas buvo simuliuojamas. Matavimo metu, buvo nupiešta 10000 skirtingų laivų nuotraukų ir matome, kad nenaudojant flyweight šablono, nuotraukos buvo nupieštos per 16s. ir buvo išnaudota 5393KB atminties. Naudojant flyweight šabloną, visas nuotraukas nupiešė per 2s. ir buvo išnaudota tik 1025KB atminties. Iš gautų rezultatų galima teigti, kad flyweight pagreitino nuotraukos piešimą

## Composite

pav. composite 6ablono diagramos fragmentas

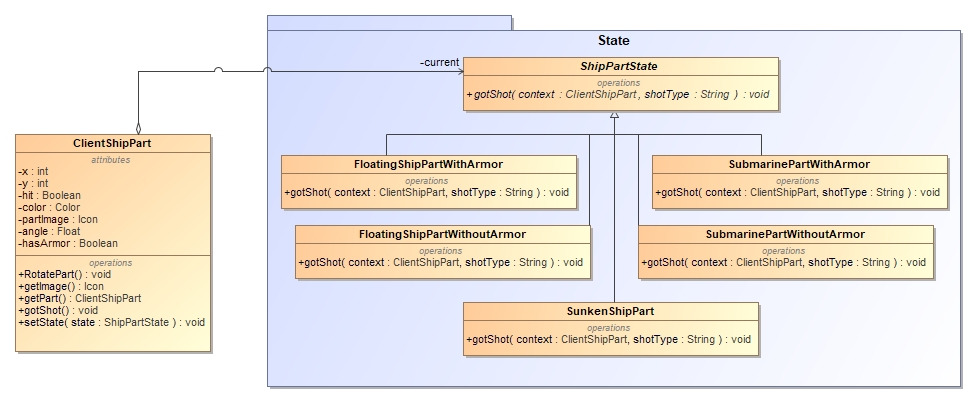


Šio šablono pagalba galime sudėtinius objektus laikyti, kaip vieną bendrinį objektą. Todėl nusprendįme šį šabloną pritaikyti prie komandos šablono, sukurdami sudėtinę komandą, kuri bus laikoma, kaip paprasta komanda, tačiau jos viduje gali būti kelios komandos arba dar daugiau kompositinių komandų, kas supaprastina komandų vykdymą. Dabar galime sukurti sudėtinę komandą, ir kai jai bus iškviesta execute() operacija, execute() operacijas atliks visos ciduje easnčios koamndos. Tai ypač aktualu, norint vienu metu atlikti dvi komandas pvz. Mūsų atveju sukurti žaidimą ir tada prie jo prisijungti.

**package** client.Models.Command.Composite;  
  
**import** client.Models.Command.Command;  
**import** client.Models.Command.CreateGameOnCommand;  
**import** client.Models.Command.JoinGameOnCommand;  
**import** client.Models.GameData;  
  
**import** java.util.ArrayList;  
  
**public class** CompositeCommand **implements** Command {  
 **private** ArrayList<Command> **children** = **new** ArrayList<>();  
  
 **public void** addChild(Command command) {  
 **children**.add(command);  
 }  
  
 **public void** removeChild(Command command) {  
 **children**.remove(command);  
 }  
  
 **public** Command getChild(**int** index) {  
 **return children**.get(index);  
 }  
  
 **public** String execute() {  
 String response = **"Error"**;  
 **for** (**var** child : **children**) {  
 **if** (child **instanceof** CreateGameOnCommand) {  
 response = child.execute();  
 }  
 **else** {  
 child.execute();  
 }  
 }  
 **return** response;  
 }  
  
 **public void** undo() {  
  
 }  
  
 **public boolean** isComposite() {  
 **return true**;  
 }  
}

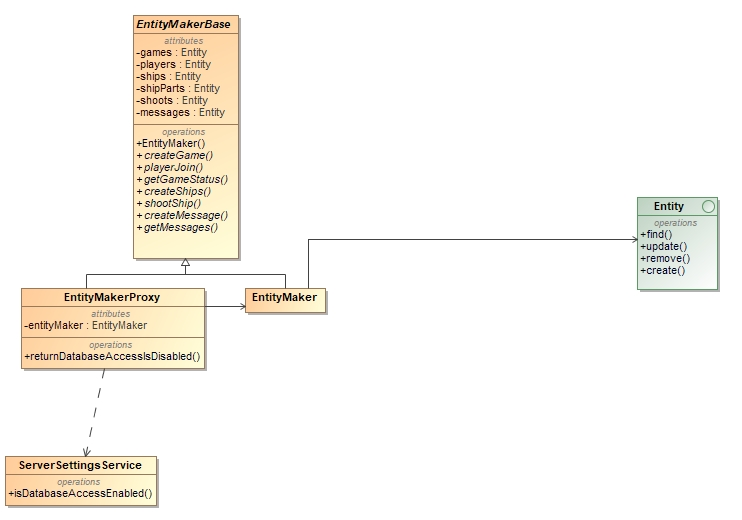
## State

pav. state šablono diagramos fragmentas



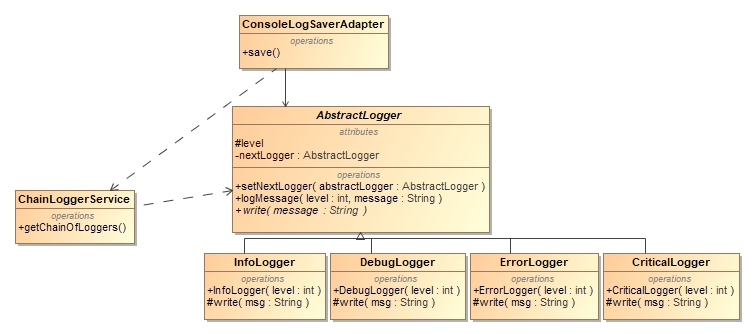
## Proxy

pav. proxy šablono diagramos fragmentas



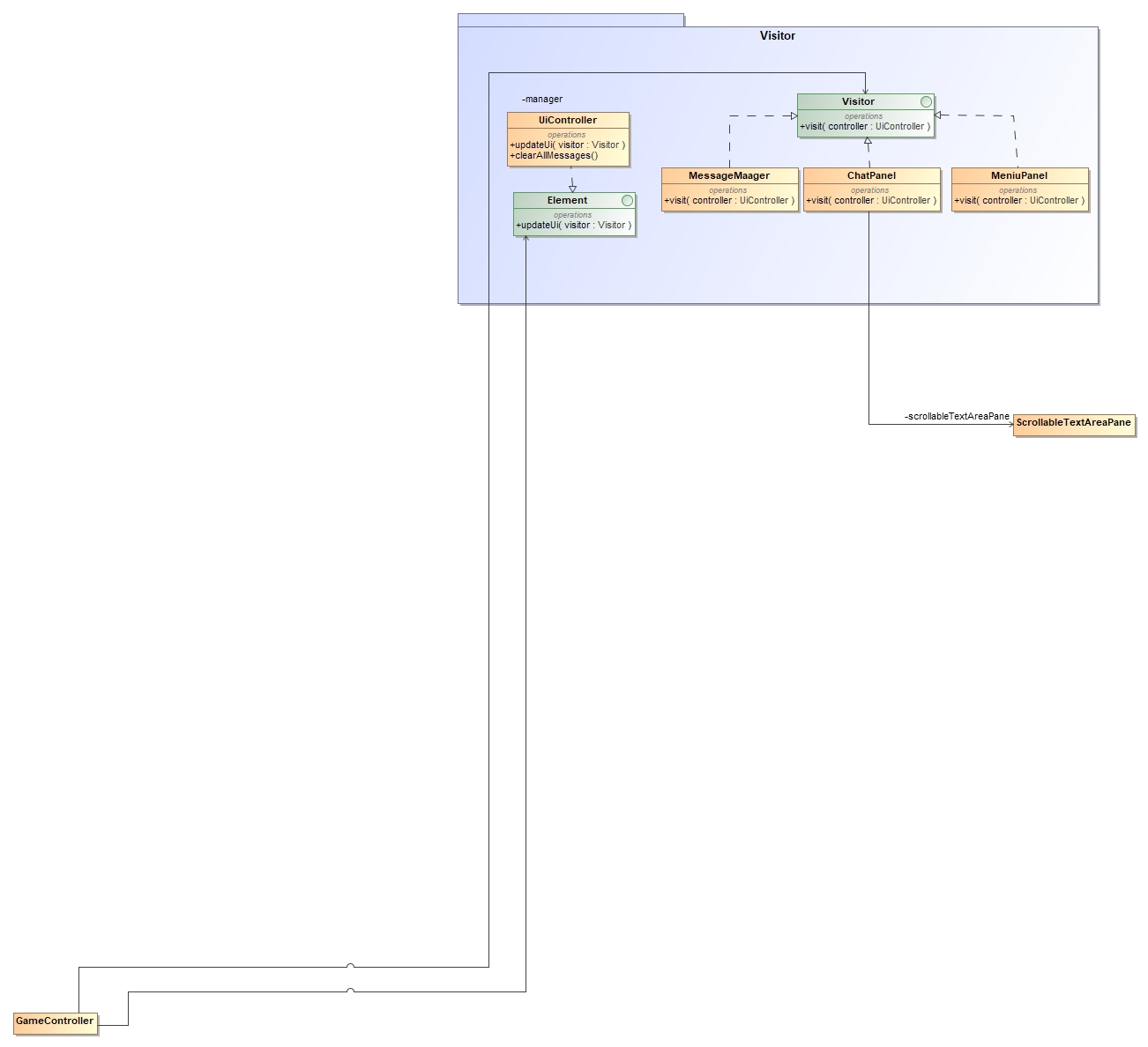
## Chain Of Responsibility

pav. chain of responsibility šablono diagramos fragmentas



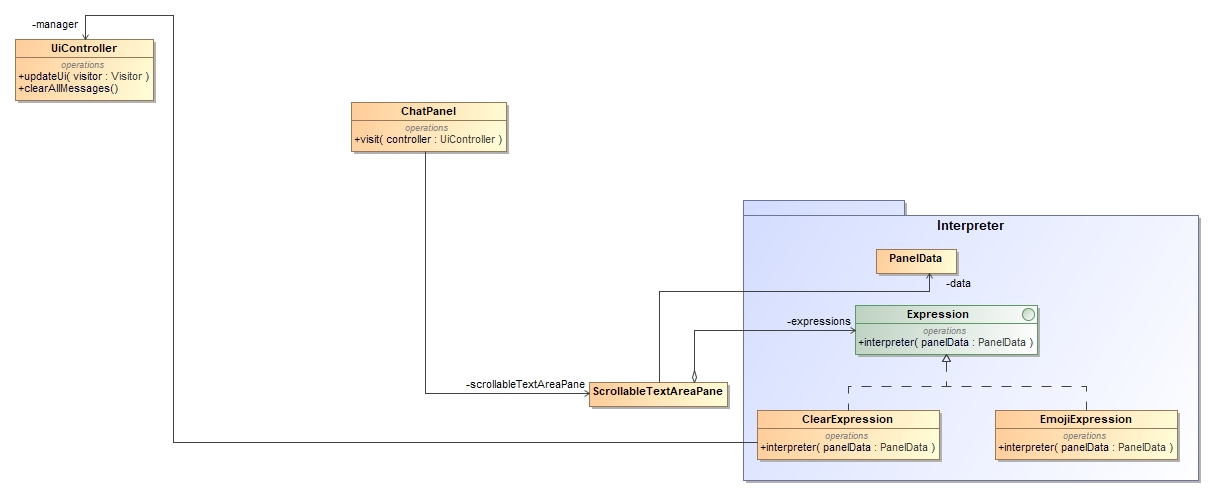
## Visitor

pav. visitor šablono šablono diagramos fragmentas



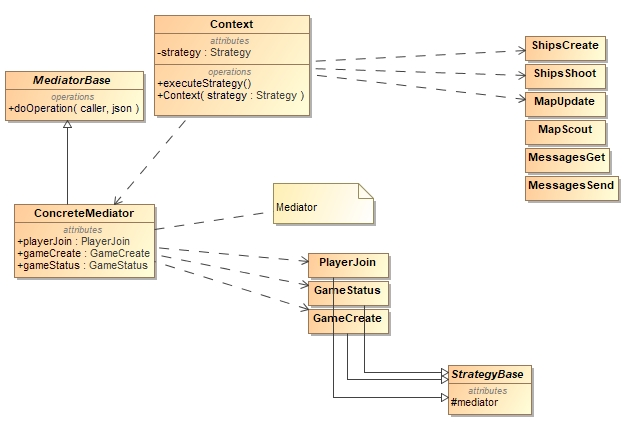
## Interpreter

pav. interpreter šablono diagramos fragmentas



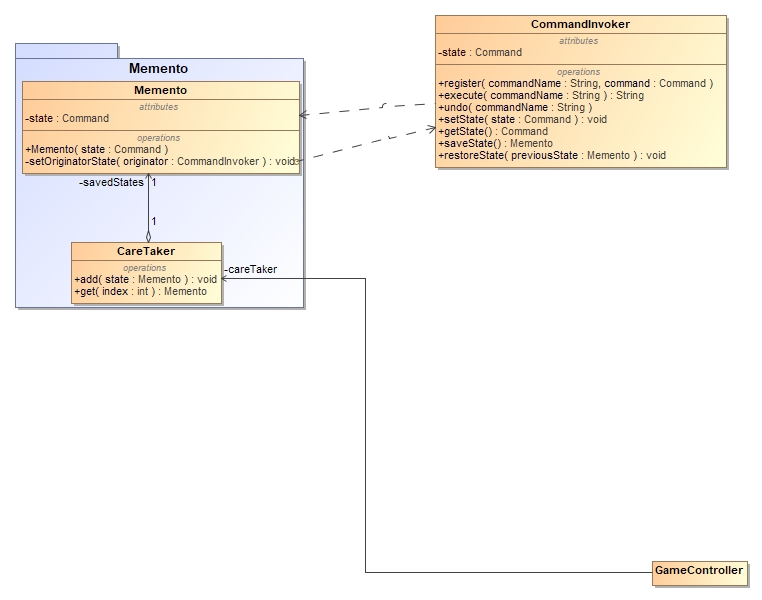
## Mediator

pav. mediator šablono diagramos fragmentas



## Memento

pav. memento šablono diagramos fragmentas



Memento šablonas yra ypač naudingas norint realizuoti undo operacijas arba sukurti kokius nors atskaitos taškus. Šis šablonas padeda išsaugoti būsenas, kurias esant reikalui galime atstatyti į praeitas. Šiuo atveju panaudojome šį šabloną komandos šablone realizuojant undo operaciją. Atlikus tam tikrą operaciją, yra sukuriamas memento objektas, kuris saugo būseną viduje, todėl iš išorės prie jos prieiti niekas negali. Būsenos atstatymas vyksta per šį sukurtą memento objektą, iškviečiant “setOriginatorState” operaciją, memento atstato būseną tik tam, kas jį sukūrė.

**package** client.Models.Command;  
  
**import** client.Models.ClientShip;  
**import** client.Views.ShipButton;  
  
**import** java.util.HashMap;  
  
**public class** CommandInvoker {  
 **private** Command **state**;  
 **private final** HashMap<String, Command> **commandMap** = **new** HashMap<>();  
  
 **public void** register(String commandName, Command command) {  
 **state** = command;  
 **commandMap**.put(commandName, command);  
 }  
  
 **public** String execute(String commandName) {  
 Command command = **commandMap**.get(commandName);  
 **if** (command == **null**) {  
 **throw new** IllegalStateException(**"Unregistered command: "** + commandName);  
 }  
 **return** command.execute();  
 }  
  
 **public void** undo(String commandName) {  
 Command command = **commandMap**.get(commandName);  
 **if** (command **instanceof** PlaceShipOnCommand) {  
 **state**.undo();  
 }  
 **else** {  
 System.***out***.println(**"Other commands does not have a functioning undo"**);  
 }  
 }  
  
 **public** ClientShip getRefShip() {  
 PlaceShipOnCommand prevState = (PlaceShipOnCommand) **state**;  
 **return** prevState.getUndoneShip();  
 }  
  
 **public** ShipButton getRefBtn() {  
 PlaceShipOnCommand prevState = (PlaceShipOnCommand) **state**;  
 **return** prevState.refShipButton();  
 }  
  
 **private void** setState(Command state) {  
 **this**.**state** = state;  
 }  
  
 **public** Memento saveState() {  
 System.***out***.println(**"Saving to memento"**);  
 **return new** Memento(**state**);  
 }  
  
 **public void** restoreState(Memento previousState) {  
 previousState.restore(**this**);  
 System.***out***.println(**"Restoring previous state"**);  
 }  
  
 **public static class** Memento {  
 **private final** Command **state**;  
  
 **public** Memento(Command state) {  
 **this**.**state** = state;  
 }  
  
 *// Only outer class can access this* **private void** restore(CommandInvoker originator) {  
 originator.setState(**state**);  
 }  
 }  
}

**package** client.Models.Memento;  
  
**import** client.Models.Command.CommandInvoker;  
  
**import** java.util.ArrayList;  
**import** java.util.List;  
  
**public class** CareTaker {  
 *// Memento states* List<CommandInvoker.Memento> **savedStates** = **new** ArrayList<>();  
  
 **public void** add(CommandInvoker.Memento state) {  
 **savedStates**.add(state);  
 }  
  
 **public** CommandInvoker.Memento get(**int** index) {  
 CommandInvoker.Memento toReturn = **savedStates**.get(index);  
 **savedStates**.remove(index);  
 **return** toReturn;  
 }  
}

# Išvados

Kadangi nusprendėme daryti žaidimą, todėl gavosi šablonų pritaikymas paprastesnis, kadangi žaidimas yra labai elastingas ir gali jo funkcionalumas būti lengvai keičiami bevystant žaidimą. Supratome, jog tokios mažos apimties projekte tiek daug šablonų apsunkina patį kodą ir sumažina kodo skaitomumą. Taip pat supratome, jog jeigu šablonai yra nepritempinėjami, o naudojami tada kada jų reikia, tai palengvina projekto vystimą ateityje.