МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний аерокосмічний університет ім. М. Є. Жуковського

«Харківський авіаційний інститут»

Факультет радіоелектроніки, комп'ютерних систем та інфокомунікацій

Кафедра комп'ютерних систем, мереж і кібербезпеки

**Лабораторна робота**

з Кросплатформенні технології

(назва дисципліни)

на тему: «Розроблення Java-застосунків з використанням об’єктноорієнтованого підходу.»

Виконав: студент 4 курсу групи № 545б

напряму підготовки (спеціальності)

123 – комп’ютерна інженерія

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(шифр і назва напряму підготовки (спеціальності))

Тимошенко О.О.

(прізвище й ініціали студента)

Прийняв: асистент каф.503

Годованюк П.А.

(посада, науковий ступінь, прізвище й ініціали)

Національна шкала: \_\_\_\_\_\_\_\_\_\_

Кількість балів: \_\_\_\_\_

Оцінка: ECTS \_\_\_\_\_

Харків – 2020

**Мета роботи:**

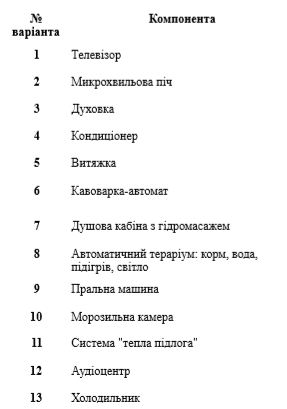
1. Відпрацювати навички об'єктно-орієнтованого аналізу і розробки архітектури програмних систем.

2. Навчитися розробляти програми на мові Java з використанням об'єктноорієнтованого підходу.Постановка завдання роботи

**Завдання**:

Описане в завданні для лабораторної роботи 4, тому не буду тут повторяти все завдання.

Мій варіант 13, тому у мене обов’язково повиненен бути холодильник.



**Виконання завдання**:

1. UML діаграма классів-пакетів



Рисунок 1 – Діаграма классів-пакетів

1. Діаграма прецедентів



Рисунок 2 – Діаграма прецедентів

**Вихідний код додатку**

Клас Component.java

package abstraction;  
  
public abstract class Component implements ITurnable  
{  
 private int id;  
 private String name;  
 private boolean isTurnedOn;  
  
 public Component(int id, String name)  
 {  
 this.id = id;  
 this.name = name;  
 }  
  
 public int getId()  
 {  
 return id;  
 }  
  
 public String getName()  
 {  
 return name;  
 }  
  
 private String getType()  
 {  
 return getClass().getName();  
 }  
  
 @Override  
 public boolean isTurnedOn()  
 {  
 return isTurnedOn;  
 }  
  
 @Override  
 public void on()  
 {  
 isTurnedOn = true;  
 }  
  
 @Override  
 public void off()  
 {  
 isTurnedOn = false;  
 }  
  
 @Override  
 public String toString()  
 {  
 return String.format("%s Component\n\tID: %d\n\tName: %s\n\tState: %s\n\t",  
 getType(), getId(), getName(), getStateString());  
 }  
  
 private String getStateString()  
 {  
 if (isTurnedOn) return "On";  
 else return "Off";  
 }  
}

Interface ITurnable.java

package abstraction;  
  
public interface ITurnable  
{  
 boolean isTurnedOn();  
  
 void on();  
 void off();  
}

Клас Recipe.java

package cases.models;  
  
public class Recipe {  
 public String name;  
  
 public String recipeText;  
  
 public Recipe(String name, String recipeText) {  
 this.name = name;  
 this.recipeText = recipeText;  
 }  
  
 @Override  
 public String toString() {  
 return "Recipe - " + name + "\n" + recipeText;  
 }  
}

Класс AirConditioner.java

package cases;  
  
import abstraction.Component;  
  
public class AirConditioner extends Component  
{  
 public AirConditioner(int id, String name)  
 {  
 super(id, name);  
 }  
}

Класс Kettle.java

package cases;  
  
import abstraction.Component;  
  
public class Kettle extends Component  
{  
 public Kettle(int id, String name)  
 {  
 super(id, name);  
 }  
}

Класс Fridge.java

package cases;  
  
import abstraction.Component;  
import cases.models.Recipe;  
  
import java.util.ArrayList;  
import java.util.List;  
  
public class Fridge extends Component  
{  
 private List<Recipe> recipesBook;  
 private int chosenRecipe;  
  
 public Fridge(int id, String name)  
 {  
 super(id, name);  
  
 super.on();  
 recipesBook = new ArrayList<Recipe>(3);  
 recipesBook.add(new Recipe("Fried potatoes", "Ingredients: 1 kg potato, oil\n" +  
 "Peel potato. Fry the potatoes for 20 minutes."));  
 recipesBook.add(new Recipe("Oatmeal with milk", "Ingredients: 100 gr oatmeal, 100 ml milk, bananas, honey or smth else\n" +  
 "Place all the ingredients into a medium microwave safe bowl and stir together.\n Heat in the microwave on high for 2 minutes."));  
 recipesBook.add(new Recipe("Baked Brie Cranberry in Bread Bowl",  
 "Ingredients: Bread boule, Cooking spray, Brie, Cranberry sauce, Thyme\n" +  
 "Recipe: First, it’s important to get a round bread boule that is bigger than a standard brie wheel.\n" +  
 "and then a lot of text))"));  
  
 chosenRecipe = 0; // chosen recipe index  
 }  
  
 public Recipe getChosenRecipe() { return recipesBook.get(chosenRecipe);}  
  
 public int getChosenRecipeId()  
 {  
 return chosenRecipe;  
 }  
  
 public String getChosenRecipeName()  
 {  
 return recipesBook.get(chosenRecipe).name;  
 }  
  
 public String getRecipes()  
 {  
 if (!isTurnedOn()) return "";  
  
 StringBuilder allRecipes = new StringBuilder();  
  
 for (int i = 0; i < recipesBook.size(); i++)  
 {  
 allRecipes.append("id:").append(i).append(" | Recipe: ").append(recipesBook.get(i)).append("\n\n");  
 }  
  
 return allRecipes.toString();  
 }  
  
 public void chooseRecipe(int id)  
 {  
 if (!isTurnedOn()) return;  
  
 if(id >= 0 && id < recipesBook.size()) {  
 chosenRecipe = id;  
 }  
 }  
  
 public String toString()  
 {  
 String selfDescription = "";  
 if (isTurnedOn())  
 {  
 selfDescription = String.format("\nChosen recipe\n\t id: %d | name: %s\n\t",  
 getChosenRecipeId(), getChosenRecipeName());  
 }  
  
 return super.toString() + selfDescription;  
 }  
  
}

Класс Lighting.java

package cases;  
  
import abstraction.Component;  
  
public class Lighting extends Component  
{  
 public Lighting(int id, String name)  
 {  
 super(id, name);  
 }  
}

Класс TV.java

package cases;  
  
import abstraction.Component;  
  
public class TV extends Component  
{  
 public TV(int id, String name)  
 {  
 super(id, name);  
 }  
}

Класс Command<TReceiver>.java

package abstraction;  
  
import java.util.Date;  
import java.util.Timer;  
import java.util.TimerTask;  
  
public abstract class Command<TReceiver> extends TimerTask  
{  
 protected IView sender;  
 protected TReceiver receiver;  
  
 private CommandQueue queue;  
 private static Thread uiUpdateDispatcher;  
  
 private Timer timer;  
  
 private Timer getTimer()  
 {  
 if (timer == null)  
 {  
 timer = new Timer();  
 }  
  
 return timer;  
 }  
  
 private void stopTimer()  
 {  
 getTimer().cancel();  
 timer = null;  
 }  
  
 public Date getExecutionTime()  
 {  
 return new Date(this.scheduledExecutionTime());  
 }  
  
 private String getType()  
 {  
 return getClass().getName();  
 }  
  
 public TReceiver getReceiver()  
 {  
 return receiver;  
 }  
  
 public IView getSender()  
 {  
 return sender;  
 }  
  
 public Command(IView sender, TReceiver receiver, CommandQueue queue)  
 {  
 this.sender = sender;  
 this.receiver = receiver;  
 this.queue = queue;  
  
 if (uiUpdateDispatcher == null)  
 {  
 uiUpdateDispatcher = new Thread(new DispatcherUpdate(sender));  
 }  
  
 queue.add(this);  
 }  
  
 public void runWithDelay(long delay)  
 {  
 if (delay > 0)  
 {  
 getTimer().schedule(this, delay);  
 }  
 else if (delay == 0)  
 {  
 run();  
 }  
 }  
  
 public void runWithDelayAndRepeat(long delay, long period)  
 {  
 getTimer().schedule(this, delay, period);  
 }  
  
 protected abstract void execute();  
  
 @Override  
 public void run()  
 {  
 if (queue.peek() == this)  
 {  
 stopTimer();  
 queue.pool();  
 execute();  
  
 uiUpdateDispatcher.run();  
 }  
 }  
  
 @Override  
 public String toString()  
 {  
 return String.format("%s\n\tTime: %s\n\tDestination: %s\n",  
 getType(), getExecutionTime().toString(), receiver.getClass().toString());  
 }  
}

Класс CommandQueue.java

package abstraction;  
  
import java.util.Comparator;  
import java.util.PriorityQueue;  
  
public class CommandQueue  
{  
 private PriorityQueue<Command> commands;  
  
 public CommandQueue()  
 {  
 Comparator<Command> commandComparator = (o1, o2) -> o2.getExecutionTime().compareTo(o1.getExecutionTime());  
  
 commands = new PriorityQueue<>(commandComparator);  
 }  
  
 public int size()  
 {  
 return commands.size();  
 }  
  
 public boolean add(Command command)  
 {  
 return commands.add(command);  
 }  
  
 public Command pool()  
 {  
 return commands.poll();  
 }  
  
 public Command peek()  
 {  
 return commands.peek();  
 }  
  
 @Override  
 public String toString()  
 {  
 StringBuilder view = new StringBuilder();  
  
 for (Command command : commands)  
 {  
 view.append(command.toString());  
 }  
  
 return view.toString();  
 }  
}

Класс DispatcherUpdate.java

package abstraction;  
  
public class DispatcherUpdate implements Runnable  
{  
 private IView view;  
  
 public DispatcherUpdate(IView view)  
 {  
 this.view = view;  
 }  
  
 @Override  
 public void run()  
 {  
 view.update();  
 }  
}

Класс FridgeChooseRecipeCommand.java

package cases;  
  
import abstraction.Command;  
import abstraction.CommandQueue;  
import abstraction.IView;  
import cases.models.Recipe;  
  
public class FridgeChooseRecipeCommand extends Command<Fridge>  
{  
 private int newRecipeId;  
  
 public FridgeChooseRecipeCommand(IView iView, Fridge fridge, int newRecipeId, CommandQueue queue)  
 {  
 super(iView, fridge, queue);  
 this.newRecipeId = newRecipeId;  
 }  
  
 @Override  
 protected void execute()  
 {  
 receiver.chooseRecipe(newRecipeId);  
 }  
}

Класс TurnOffCommand.java

package cases;  
  
import abstraction.\*;  
  
public class TurnOffCommand extends Command<ITurnable>  
{  
 public TurnOffCommand(IView iView, ITurnable iTurnable, CommandQueue queue)  
 {  
 super(iView, iTurnable, queue);  
 }  
  
 @Override  
 protected void execute()  
 {  
 receiver.off();  
 }  
}

Класс TurnOnCommand.java

package cases;  
  
import abstraction.\*;  
  
public class TurnOnCommand extends Command<ITurnable>  
{  
 public TurnOnCommand(IView iView, ITurnable iTurnable, CommandQueue queue)  
 {  
 super(iView, iTurnable, queue);  
 }  
  
 @Override  
 protected void execute()  
 {  
 receiver.on();  
 }  
}

Класс ComponentFactory.java

package abstraction;  
  
import java.util.concurrent.atomic.AtomicInteger;  
  
public abstract class ComponentFactory  
{  
 private static final AtomicInteger autoIncrementId = new AtomicInteger(0);  
  
 protected int getAutoIncrementId()  
 {  
 return autoIncrementId.getAndIncrement();  
 }  
  
 public Component createComponent(String name)  
 {  
 return create(name);  
 }  
  
 protected abstract Component create(String name);  
}

Класс ComponentConnector.java

package connections;  
  
import abstraction.Component;  
import abstraction.ComponentFactory;  
  
import java.util.Collection;  
import java.util.HashMap;  
import java.util.Map;  
  
public class ComponentConnector  
{  
 private Map<Integer, Component> components;  
  
 public ComponentConnector()  
 {  
 components = new HashMap<>();  
 }  
  
 public boolean connect(Component component)  
 {  
 if (!hasIndex(component.getId()))  
 {  
 components.put(component.getId(), component);  
 return true;  
 }  
 else return false;  
 }  
  
 public boolean create(String name, ComponentFactory factory)  
 {  
 return connect(factory.createComponent(name));  
 }  
  
 public boolean remove(Integer id)  
 {  
 if (hasIndex(id))  
 {  
 components.remove(id);  
 return true;  
 }  
 else return false;  
 }  
  
 public Collection<Component> getComponents()  
 {  
 return components.values();  
 }  
  
 public Component get(Integer id)  
 {  
 return components.get(id);  
 }  
  
 public boolean hasIndex(Integer index)  
 {  
 return components.containsKey(index);  
 }  
  
 public String getComponentsTextView()  
 {  
 StringBuilder view = new StringBuilder();  
  
 for (Component component : getComponents())  
 {  
 view.append(component.toString());  
 }  
  
 return view.toString();  
 }  
}

Перечислення ComponentType.java

package connections;  
  
public enum ComponentType  
{  
 AirConditioner,  
 Fridge,  
 Kettle,  
 Lighting,  
 TV  
}

Класс AirConditionerFactory.java

package factories;  
  
import abstraction.Component;  
import abstraction.ComponentFactory;  
import cases.AirConditioner;  
  
public class AirConditionerFactory extends ComponentFactory  
{  
 @Override  
 protected Component create(String name)  
 {  
 return new AirConditioner(getAutoIncrementId(), name);  
 }  
}

Класс FridgeFactory.java

package factories;  
  
import abstraction.Component;  
import abstraction.ComponentFactory;  
import cases.Fridge;  
  
public class FridgeFactory extends ComponentFactory  
{  
 @Override  
 protected Component create(String name)  
 {  
 return new Fridge(getAutoIncrementId(), name);  
 }  
}

Класс KettleFactory.java

package factories;  
  
import abstraction.Component;  
import abstraction.ComponentFactory;  
import cases.Kettle;  
  
public class KettleFactory extends ComponentFactory  
{  
 @Override  
 protected Component create(String name)  
 {  
 return new Kettle(getAutoIncrementId(), name);  
 }  
}

Класс LightingFactory.java

package factories;  
  
import abstraction.Component;  
import abstraction.ComponentFactory;  
import cases.Lighting;  
  
public class LightingFactory extends ComponentFactory  
{  
 @Override  
 protected Component create(String name)  
 {  
 return new Lighting(getAutoIncrementId(), name);  
 }  
}

Класс TVFactory.java

package factories;  
  
import abstraction.Component;  
import abstraction.ComponentFactory;  
import cases.TV;  
  
public class TVFactory extends ComponentFactory  
{  
 @Override  
 protected Component create(String name)  
 {  
 return new TV(getAutoIncrementId(), name);  
 }  
}

Інтерфейс IView.java

package abstraction;  
  
import java.io.PrintStream;  
  
public interface IView  
{  
 void update();  
}

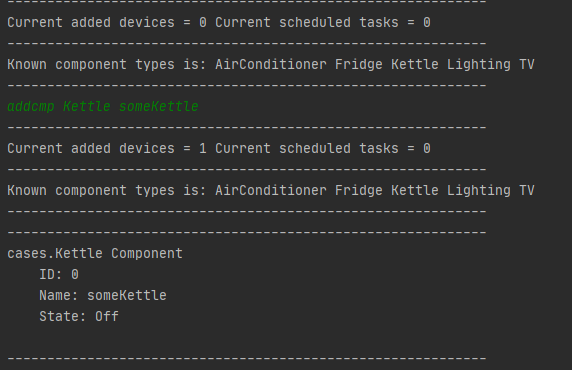
Класс ControlConsole.java

package console;  
  
import abstraction.\*;  
import cases.\*;  
import connections.ComponentConnector;  
import connections.ComponentType;  
import factories.\*;  
  
import java.io.\*;  
import java.text.ParseException;  
import java.text.SimpleDateFormat;  
import java.util.\*;  
  
public class ControlConsole implements IView  
{  
 private BufferedReader cin;  
 private PrintStream cout;  
 private ComponentConnector componentConnector;  
 private CommandQueue commandQueue;  
 private boolean helpOn;  
  
 public ControlConsole(InputStream cin, PrintStream cout, ComponentConnector componentConnector)  
 {  
 this.cin = new BufferedReader(new InputStreamReader(cin));  
 this.cout = cout;  
 this.componentConnector = componentConnector;  
 this.commandQueue = new CommandQueue();  
 helpOn = false;  
 }  
  
 @Override  
 public void update()  
 {  
 printLine();  
 printHeader();  
 printLine();  
  
 printKnownTypes();  
 printLine();  
 if (componentConnector.getComponents().size() != 0)  
 {  
 printLine();  
 cout.println(componentConnector.getComponentsTextView());  
 printLine();  
 }  
 if (commandQueue.size() != 0)  
 {  
 printTimers();  
 printLine();  
 }  
 if (helpOn)  
 {  
 printUserCommands();  
 }  
  
 try  
 {  
 captureInput();  
 }  
 catch (IOException e)  
 {  
 e.printStackTrace();  
 }  
 }  
  
 private void printUserCommands()  
 {  
 cout.println(help);  
 }  
  
 private void printKnownTypes()  
 {  
 StringBuilder types = new StringBuilder();  
 types.append("Known component types is: ");  
  
 for (ComponentType type: ComponentType.values())  
 {  
 types.append(type).append(" ");  
 }  
  
 cout.println(types.toString());  
 }  
  
 private void printTimers()  
 {  
 cout.println(commandQueue.toString());  
 }  
  
 private void printLine()  
 {  
 cout.println("------------------------------------------------------------");  
 }  
  
 private void printHeader()  
 {  
 cout.println(String.format("Current added devices = %d " +  
 "Current scheduled tasks = %d", componentConnector.getComponents().size(), commandQueue.size()));  
 }  
  
 private static final String help = "Manipulation commands:\n" +  
 "[addcmp \*type\* \*name\*] - Adds component of specified type with specified name\n" +  
 "[rmcmp \*id\*] - Removes component with specified id\n" +  
 "[tron [-t HH:mm:ss] \*id\*] - Turns on component on specified time\n" +  
 "[trof [-t HH:mm:ss] \*id\*] - Turns off component on specified time" +  
 "Fridge commands:\n" +  
 "[frselrec \*id\_fridge\* \*id\_recipe\* - Select recipe by id\n" +  
 "[frgetcurrec \*id\_fridge\* - Get recipe for selected fridge\n" +  
 "[frreciptslist \*id\_fridge\* - List of recipes for the selected fridge\n" +  
 "[help] - Show/hide this text on every action";  
  
 private void captureInput() throws IOException  
 {  
 String userCommand = cin.readLine();  
 String delimiter = " ";  
 ArrayList<String> signatures = new ArrayList<>(Arrays.asList(userCommand.split(delimiter)));  
  
 int delayKeyPosition = signatures.indexOf("-t");  
 long delay = 0;  
 try  
 {  
 if (delayKeyPosition != -1)  
 {  
 delay = parseDelay(signatures, delayKeyPosition);  
  
 signatures.subList(delayKeyPosition, delayKeyPosition + 2).clear();  
 }  
 }  
 catch (IOException e)  
 {  
 printWaitForInputAndUpdate("Wrong delay signature. Delay signature must be [-t HH:mm:ss]");  
 }  
  
 String commandType = signatures.get(0);  
 signatures.remove(0);  
  
 Command command = null;  
  
 switch (commandType)  
 {  
 case "addcmp" :  
 {  
 createComponentCommand(signatures);  
 break;  
 }  
 case "rmcmp" :  
 {  
 removeComponentCommand(signatures);  
 break;  
 }  
 case "tron" :  
 {  
 command = createSwitcherCommand(signatures, true);  
 break;  
 }  
 case "trof" :  
 {  
 command = createSwitcherCommand(signatures, false);  
 break;  
 }  
 case "frselrec" : // select by id  
 {  
 command = createFridgeCommand(signatures);  
 break;  
 }  
 case "frgetcurrec" : // get current recipe  
 {  
 getCurrentRecipe(signatures);  
 break;  
 }  
 case "frreciptslist" :  
 {  
 getRecipesList(signatures);  
 break;  
 }  
 case "help" :  
 {  
 helpOn = !helpOn;  
 break;  
 }  
 default :  
 {  
 printWaitForInputAndUpdate("Unknown Command! Type \*help\* to display full commands list.");  
 break;  
 }  
 }  
  
 if (command != null)  
 {  
 command.runWithDelay(delay);  
 update();  
 }  
 }  
  
 private void createComponentCommand(ArrayList<String> signatures)  
 {  
 ComponentFactory factory = null;  
 String type = signatures.get(0);  
 String name = signatures.get(1);  
  
 if (type.equals(ComponentType.AirConditioner.toString()))  
 {  
 factory = new AirConditionerFactory();  
 }  
 else if (type.equals(ComponentType.Kettle.toString()))  
 {  
 factory = new KettleFactory();  
 }  
 else if (type.equals(ComponentType.Fridge.toString()))  
 {  
 factory = new FridgeFactory();  
 }  
 else if (type.equals(ComponentType.Lighting.toString()))  
 {  
 factory = new LightingFactory();  
 }  
 else if (type.equals(ComponentType.TV.toString()))  
 {  
 factory = new TVFactory();  
 }  
  
 if (factory != null)  
 {  
 componentConnector.create(name, factory);  
 update();  
 }  
 else  
 {  
 printWaitForInputAndUpdate("Wrong addcmp command signature. Must be > addcmp \*type\* \*name\*");  
 }  
 }  
  
 private void removeComponentCommand(ArrayList<String> signatures)  
 {  
 Integer id = parseId(signatures.get(0));  
  
 if (id == null || !componentConnector.remove(id))  
 {  
 printWaitForInputAndUpdate("Index value \*" + signatures.get(0) + "\* is not exist.");  
 }  
 else update();  
 }  
  
 private Command createSwitcherCommand(ArrayList<String> signatures, boolean state)  
 {  
 Integer id = parseId(signatures.get(0));  
 Command command;  
  
 if (id != null & componentConnector.hasIndex(id))  
 {  
 Component receiver = componentConnector.get(id);  
 if (state)  
 {  
 command = new TurnOnCommand(this, receiver, commandQueue);  
 }  
 else  
 {  
 command = new TurnOffCommand(this, receiver, commandQueue);  
 }  
 }  
 else  
 {  
 printWaitForInputAndUpdate("Index value \*" + id + "\* is not exist in component connector");  
 command = null;  
 }  
  
 return command;  
 }  
  
 private Command createFridgeCommand(ArrayList<String> signatures)  
 {  
 Integer id = parseId(signatures.get(0));  
 Command command;  
  
 if (id != null & componentConnector.hasIndex(id)) {  
 Component receiver = componentConnector.get(id);  
 if ((receiver instanceof Fridge)) {  
 if (signatures.size() >= 2) {  
 Integer recipeId = parseId(signatures.get(1));  
 command = new FridgeChooseRecipeCommand(this, (Fridge) receiver,  
 recipeId, commandQueue);  
 }  
 else {  
 printWaitForInputAndUpdate("Incorrect ID of new recipe");  
 command = null;  
 }  
 }  
 else {  
 printWaitForInputAndUpdate("Element with index " + id + "is not a Fridge");  
 command = null;  
 }  
 }  
 else {  
 printWaitForInputAndUpdate("Index value \*" + id + "\* is not exist in component connector");  
 command = null;  
 }  
  
 return command;  
 }  
  
 private void getCurrentRecipe(ArrayList<String> signatures)  
 {  
 Integer id = parseId(signatures.get(0));  
  
 if (id != null & componentConnector.hasIndex(id))  
 {  
 Component receiver = componentConnector.get(id);  
 if ((receiver instanceof Fridge))  
 {  
 printWaitForInputAndUpdate(((Fridge) receiver).getChosenRecipe().toString());  
 }  
 else  
 {  
 cout.println("Element with index " + id + "is not a Fridge");  
 }  
 }  
 else  
 {  
 printWaitForInputAndUpdate("Index value \*" + id + "\* is not exist in component connector");  
 }  
 }  
 private void getRecipesList(ArrayList<String> signatures)  
 {  
 Integer id = parseId(signatures.get(0));  
  
 if (id != null & componentConnector.hasIndex(id))  
 {  
 Component receiver = componentConnector.get(id);  
 if ((receiver instanceof Fridge))  
 {  
 printWaitForInputAndUpdate(((Fridge) receiver).getRecipes());  
 }  
 else  
 {  
 cout.println("Element with index " + id + "is not a Fridge");  
 }  
 }  
 else  
 {  
 printWaitForInputAndUpdate("Index value \*" + id + "\* is not exist in component connector");  
 }  
 }  
  
  
 private Integer parseId(String signature)  
 {  
 try  
 {  
 int index = Integer.parseInt(signature);  
 if (index < 0)  
 {  
 throw new NumberFormatException();  
 }  
 else return index;  
 }  
 catch (NumberFormatException e)  
 {  
 printWaitForInputAndUpdate("\*id\* must be an integer positive value");  
 return null;  
 }  
 }  
  
 private long parseDelay(ArrayList<String> signatures, int delayKeyPosition) throws IOException  
 {  
 String time = signatures.get(delayKeyPosition + 1);  
 SimpleDateFormat parser = new SimpleDateFormat("HH:mm:ss");  
  
 try  
 {  
 Calendar runTime = new GregorianCalendar();  
 runTime.setTime(parser.parse(time));  
  
 Calendar fullRunTime = Calendar.getInstance();  
 fullRunTime.set(Calendar.HOUR, runTime.get(Calendar.HOUR));  
 fullRunTime.set(Calendar.MINUTE, runTime.get(Calendar.MINUTE));  
 fullRunTime.set(Calendar.SECOND, runTime.get(Calendar.SECOND));  
  
 return fullRunTime.getTimeInMillis() - System.currentTimeMillis();  
 }  
 catch (ParseException e)  
 {  
 throw new IOException();  
 }  
 }  
  
 private void printWaitForInputAndUpdate(String string)  
 {  
 cout.println(string);  
 waitForUserInput();  
 update();  
 }  
  
 private void waitForUserInput()  
 {  
 cout.println("Press enter to continue...");  
  
 try  
 {  
 cin.readLine();  
 }  
 catch (IOException e)  
 {  
 e.printStackTrace();  
 }  
 }  
}

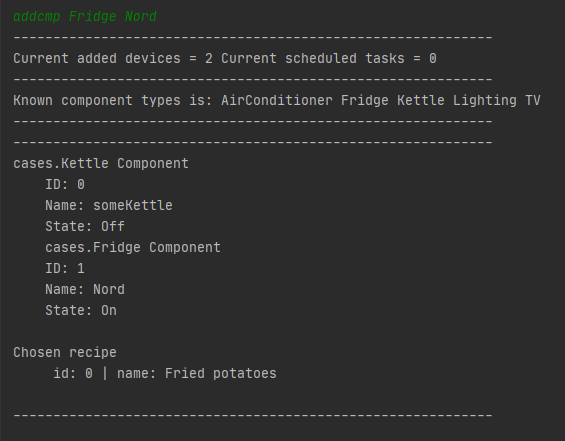
Класс Main.java

package console;  
  
import connections.ComponentConnector;  
  
public class Main  
{  
 public static void main(String[] args)  
 {  
 ComponentConnector componentConnector = new ComponentConnector();  
 ControlConsole console = new ControlConsole(System.in, System.out, componentConnector);  
  
 console.update();  
 }  
}

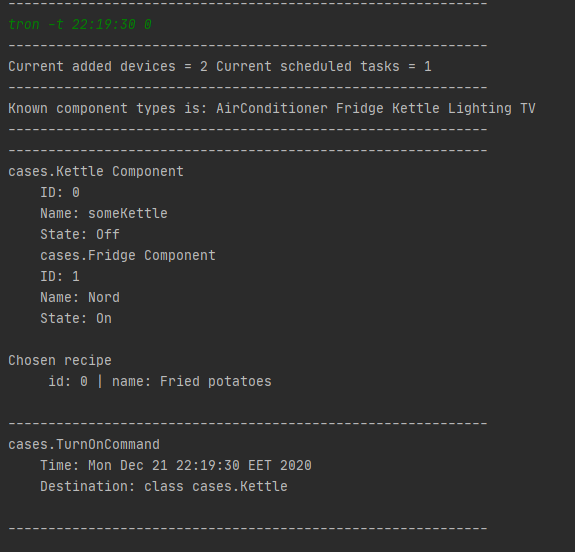
1. **Результати**



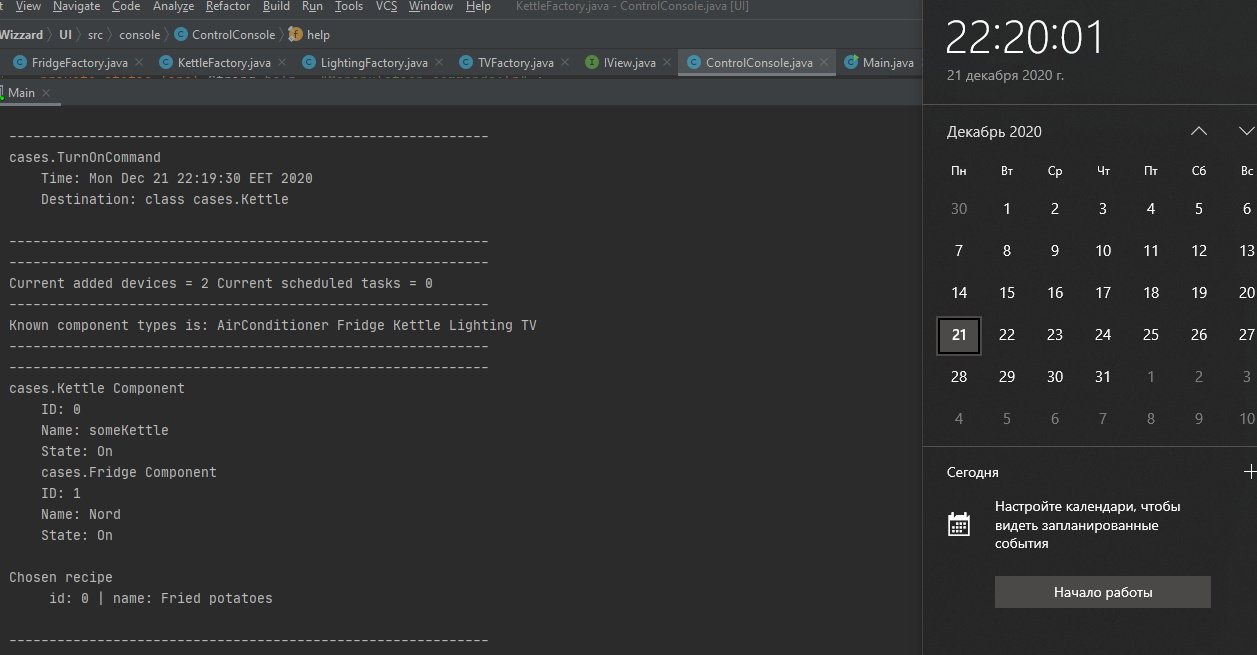
Додавання чайника



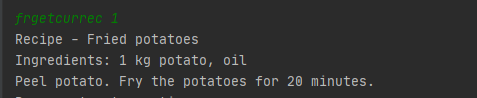
Додавання холодильника



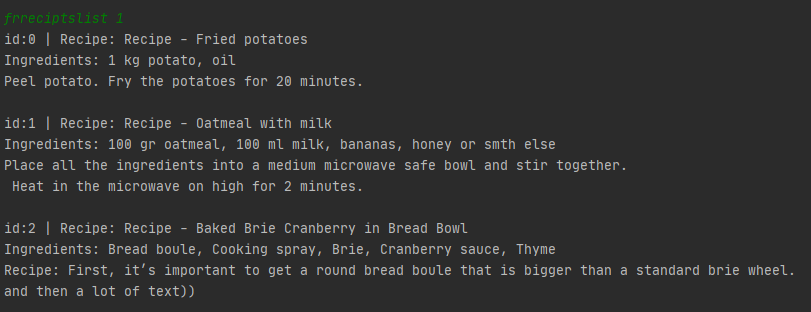
Планування задачі (включити чайник)



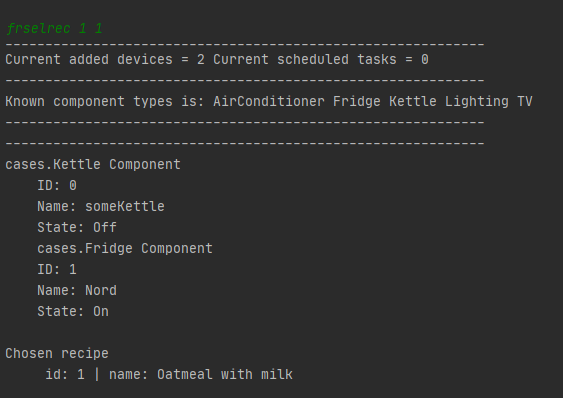
Виконання запланованої задачі (включений чайник)



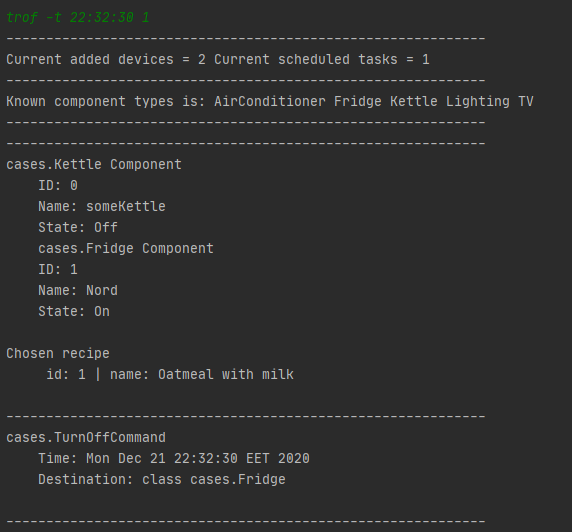
Отримання рецепту, який є обраним для холодильника з ID = 1



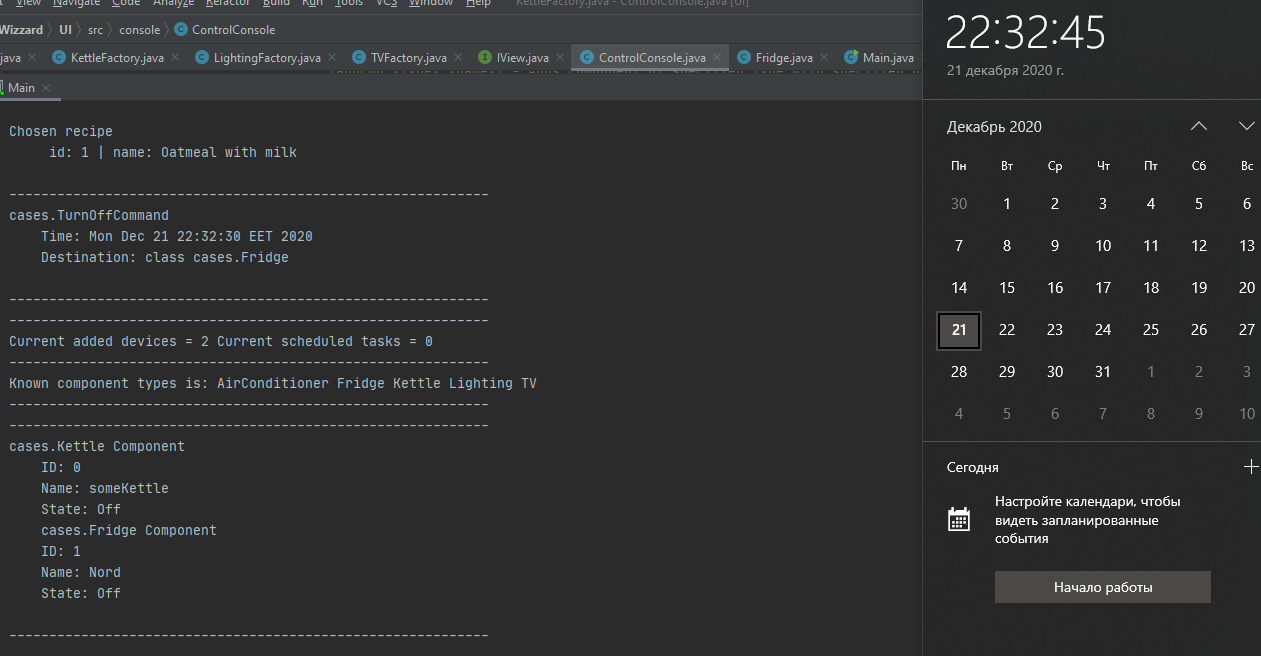
Вивести всі доступні рецепти для заданого холодильника



Обрання іншого рецепту



Планування задачі вимкнення холодильника



Вимкнення холодильника в запланований час

1. Посилання на репозитарій із проєктом у системі керування версіями.

<https://github.com/OlehTymoshenko/Cross-platform-technologies>

**Висновки:**

В результаті виконання лабораторної роботи я розробив Java-застосунок з використанням ОО підходу. При розробці програми використав патерни – команда і фабричний метод.