Академија струковних
студија Шумадија
Одсек Крагујевац

Студијски програм: Информатика
Предмет: Објектно оријентисано програмирање

Пројектни задатак - Базен

Предметни наставник:
др Александар Мишковић

Студент:
Магдалена Јакшић, 23/2020

Крагујевац 2022.

Садржај

1. Увод

Објектно оријентисано програмирање је стил програмирања који ради са објектима, елементима програма који имају своја стања и понашање. Програм се реализује кроз мењање стања објеката. Акценат се ставља на класе. Сваки пут када се креира објекат, програм прво гледа класу, па се прави објекат према правилима која су дефинисана у тој класи. Тако се креирају више различитих објеката истог типа, истих или различитих вредности атрибута које поседују.

У овом раду ће преко практичног примера у IntelliJ IDEA Community Edition бити приказана објектно оријентисана структура програма и њени основни принципи.

2. Основни принципи објектно оријентисаног програмирања

Објектно оријентисано програмирање базира се на класама и објектима. У основне концепте поред класе и објекта, спадају полиморфизам, апстракција, енкапсулација и наслеђивање.

2.1. Класа

Класа садржи поља са атрибутима и методе. Поља дефинишу особине објеката и током извршавања програма особине се мењају, чиме се мења и комплетно унутрашње стање објекта. Методе чине функционалност и то су скупови наредби које утичу на поља како би се променило унутрашње стање објеката. Декларише се помоћу резервисане речи class.

2.2. Објекат

Објекат представља инстанцу, примерак, своје класе. Поседује све особине и понашања које та класа дефинише. Има своје карактеристике које су дефинисане од стране класе и индентитет у складу са постављеним начелима класе.

2.3. Апстракција

Апстракција подразумева поједностављивање карактеристика неке класе да би се нагласиле важне особине, а игнорисале небитне у зависности од потреба. Бирају се особине којима класа дефинише неки објекат. Одређени детаљи се занемарују и узимају се заједничке карактеристике класе, у зависности од потребе датог програма.

2.4. Енкапсулација

Енкапсулација је процес где се информација о класи штити од директног приступа. Једини начин приступа је путем дефинисаних метода. Скривају се детаљи, корисник само зна шта може да ради са одређеним објектом. Смањује се шанса да дође до грешке и процес се спроводи тако што се чланови класе деле на јавне и приватне.

Јавни чланови се користе у било ком делу програма, док је приватним могуће приступити само у оквиру дефинисане класе и тако се креира одређена заштита која штити чланове. Ипак сви чланови у оквиру тела класе су међусобно доступни.

2.5. Наслеђивање

Наслеђивање подразумева идентификацију заједничких особина различитих класа и отклања потребу да се исти атрибути пишу неколико пута. На тај начин једна класа може да наследи карактеристике друге класе.

По том концепту може се дефинисати класна хијерархија. Дефинише се начин на који се може извести нови објекат. Изведена класа је потклаcа а општија је наткласа и чине хијерархију класа.

2.6. Полиморфизам

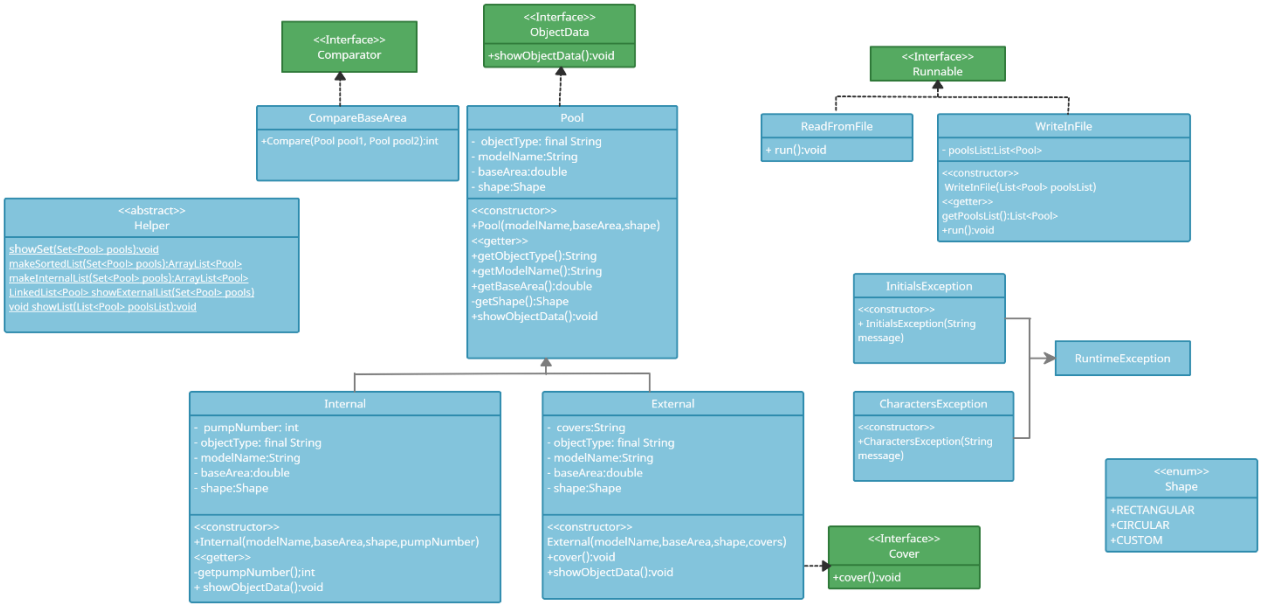
Полиморфизам омогућава да основна класа дефинише функције које ће бити заједничке за све изведене класе које могу на свој начин да имплементирају те функције. Основна и изведене класе формирају хијерархију и основна класа поседује све што изведене класе могу да користе. Изведене класе на овај начин имају могућност да самостално имплементирају функције.

3. Пројектни задатак - Базен

Циљ пројектног задатка је направити конзолну апликацију за потребе грађевинске фирме, која ће садржати податке о базенима и могућност читања и уписивања листе у текстуалну датотеку. Детаљна поставка задатка је дата у поглављу 4.

3.1. Структура пројекта помоћу UML дијаграма

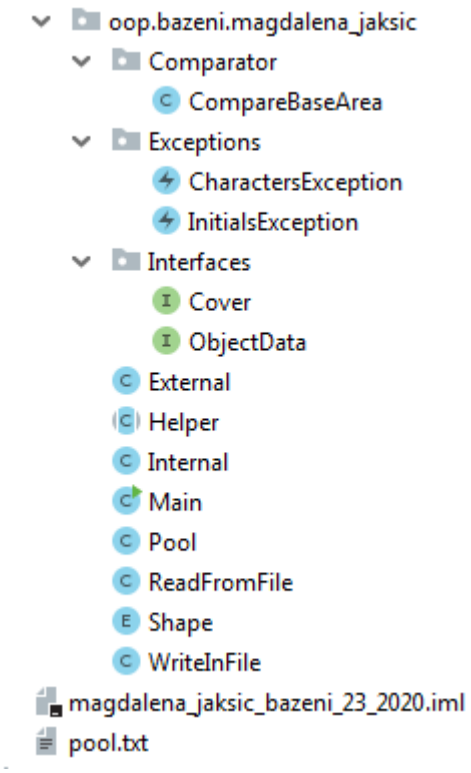
UML класни дијаграм се користи у софтверском инжењерству за опис структуре система приказујући класе са њиховим пољима и методама и међусобним релацијама. Зелени објекти у примеру приказују интерфејсе док плави приказују класе. Испрекидане стрелице означавају имплементацију интерфејса у класи док пуне означавају наслеђивање и иду од поткласе ка наткласи.



Према дијаграму, класе Internal и External наслеђују класу Pool, која дефинише њихове заједничке атрибуте и имплементира интерфејс ObjectData и дефинише његову методу showObjectData() тако да исписује заједничке атрибуте базена. У својој структури класе Internal и External наслеђују методу showObjectData() од своје наткласе Pool и редефинишу је тако да се поред већ дефинисаног исписа исписују и њихови остали атрибути. Класа External имплементира интерфејс Cover и дефинише његову методу cover() тако да се на основу одређеног уноса исписује редефинисана порука о могућности наткривања базена.

Класе InitialsException и CharactersException наслеђују Јавину класу RuntimeException и служе за приказ поруке о грешци при неадекватном дефинисању исписа података о базенима. Класа CompareBaseArea имплементира Јавин Comparator интерфејс и служи за поређење два објекта, базена, по површини основе. Enum класа Shape дефинише константе у смислу могућих облика базена. Класе WriteInFile и ReadFromFile имплементирају Јавин интерфејс Runnable и служе за упис одређене листе базена у текстуалну датотеку, као и читање, испис, из ње. Апстрактна класа Helper садржи статичке методе које се позивају у Main класи и које служе са рад са листама.

3.2.Структура пројекта у IntelIIJ IDEA Community Edition



3.3.Интерфејси

Интерфејс је потпуно апстракна класа која се користи за груписање повезаних метода са празним телима. Да би се приступило методи интерфејса мора да буде имплементиран од стране друге класе са кључном речи implements, која дефинише тело методе интерфејса. Приликом имплементације интерфејса и дефинисања тела методе мора се користити анотација @override. Подразумеване вредности метода су public и abstract, а атрибута public, static и final.

3.3.1. Интерфејс Cover

```
        //INTERFACE FOR INFORMATION ABOUT EXTERNAL POOL COVERS
public interface Cover {
   void cover();
}
```

3.3.2. Интерфејс ObjectData

```
        //INTERFACE FOR SHOWING OBJECT DATA
public interface ObjectData {
   void showObjectData();
}
```

3.4.Изузеци

Изузетак је догађај који се појављује приликом извршења програма и прекида нормалан ток програма. То је генерално нежељени или неочекивани догађај који може да се појави у време комајлирања или извршења програма. Управљање изузетцима омогућава решавање проблема приликом њиховог настанка . Могу се поделити према хијерархији класа и према врсти. Према хијерархији класа се деле на класу Exception и класу Error. Према врсти се деле на проверљиве, непроверљиве и грешке. Проверљиве проверава компајлер у тренутку писања кода и њима припадају инстанце класе Exception осим RuntimeException. Непроверљиве компајлер не може проверити и обично су грешке које програмер сам направи. Њима припадају инстанце класе RuntimeException. Грешке су проблеми настали током извршења програма и њима припадају инстанце класе Error.

У примеру дефинисана су два изузетака InitialsException и CharactersException. Садрже конструктор који прима поруку као аргумент. Наслеђују Јавину класу RuntimeException која је суперкласа изузетака који могу бити избачени приликом нормалног извршавања Јавине виртуелне машине. Све потклассе класе RuntimeException припадају непровереним изузетцима.

3.4.1. InitialsException

```
        //INITIALS EXCEPTION FOR OUTPUT MESSAGE
public class InitialsException extends RuntimeException {
   public InitialsException(String message) {
      super(message);
   }
}
```

3.4.2. CharactersException

```
//CHARACTER EXCEPTION FOR OUTPUT MESSAGE
public class CharactersException extends RuntimeException {
   public CharactersException(String message) {
      super(message);
   }
}
```

3.5. Компаратор

Компаратор је интерфејс у Јави који служи за сортирање објеката. Дефинисан је у java.util пакету. Има две методе compare(Object obj1,Object obj2) и equals(Object element). Прва се користи за поређење два објекта. Враћа позитиван број ако је први објекат већи од другог, 0 ако су једнаки и негативан број ако је први објекат мањи.

3.5.1. CompareBaseArea

```
//COMPARING BASE AREA OF POOLS
public class CompareBaseArea  implements Comparator<Pool> {
   @Override
   public int compare(Pool pool1, Pool pool2) {
      if(Double.compare(pool1.getBaseArea(),pool2.getBaseArea())==0){
         return 1;
      } else if(Double.compare(pool1.getBaseArea(),pool2.getBaseArea())<0){
         return -1;
      } else {return  0;}
   }
}
```

Класа имплементира Јавин интерфејс Comparator и служи за поређење два објекта по површини основе.

3.6. Класе

3.6.1. Класа Pool

Класа Pool је наткласа класама Internal и External. Однос између наведене три класе приказује један од основних принципа објектно оријентисаног програмирања - наслеђивање. Декларише њихове заједничке варијабле у које спадају име модела, површина основе и облик базена. Такође дефинише константу која даје податак о типу објекта. Кључна реч private је модификатор приступа и означава да се атрибуту може приступити само унутар декларисане класе. Генерише конструктор који служи за иницијализацију објекта и њему се прослеђују параметри који служе за иницијализацију атрибута.

```
//CONSTANT FOR OBJECT TYPE INTERNAL/EXTERNAL
private final String objectType = getClass() == Internal.class ? "Internal" : "External";

//CONSTRUCTOR FOR COMMON PROPERTIES FOR OBJECTS
private String modelName;
private double baseArea;
private Shape shape;

public Pool(String modelName, double baseArea, Shape shape) {
   this.modelName = modelName;
   this.baseArea = baseArea;
   this.shape = shape;
}
```

Приватним атрибутима је могуће приступити уз помоћ јавне get методе која враћа вредност атрибута. Пошто класа садржи само get методу, класни атрибути су read-only.

```
//GETTING VALUES OF COMMON PROPERTIES
String getObjectType() {
  return objectType;
}

String getModelName() {
   return modelName;
}

public double getBaseArea() {
   return baseArea;
}
private Shape getShape() {   return shape; }
```

Класа Pool имплементира интерфејс ObjectData који садржи методу showObjectData() и дефинише је тако да врши испис иницијала и заједничких атрибута по задатим правилима задатка.

```
public class Pool implements ObjectData

        ...

            @Override
   public void showObjectData() {
      String initials = "[MJ]";
      String commonData = "- type: " + getObjectType() + " - name: " + getModelName() + " -base area: " + getBaseArea()
+ " -shape: " + getShape();
      if (initials.length() != 4 || initials.charAt(0) != '[' || initials.charAt(1) != 'M' || initials.charAt(2) != 'J' ||
initials.charAt(3) != ']') {
         throw new InitialsException("Initials are not in the right format![MJ]");
      } else if(commonData.contains("{") || commonData.contains("}")||  commonData.contains("=") ) {
         throw new CharactersException("\"You tried to use \\\"{\\\", \\\"}\\\" and/or \\\"=\\\" in presentation
message!\"");
      } else {
         System.out.print(initials+commonData+" ");
      }
   }
}
```

Кључна реч throw служи да избаци изузетак из методе или било ког блока кода. Могу се избацити проверени или непроверени изузетци. Уобичајено се користи да избаци прављене изуетке. У примеру прављени изузетци су InitialsException и CharacterException који се избацују приликом неадекватног дефинисања исписа и као аргумент им се прослеђује одговарајућа порука.

3.6.2. Класа External

External класа је поткласа класе Pool, наслеђује дефинисане атрибуте и методе. Поред заједничких атрибута има свој јединствен атрибут covers који служи за давање информације о могућности покривања базена. Садржи конструктор коме се прослеђују наслеђени атрибути и додаје свој. Private је модификатор приступа и означава да је променљива видљива само унутар класе у којој је дефинисана.

```
public class External extends Pool implements Cover

...

// INPUT AVAILABILITY OF COVERS FOR EXTERNAL POOL
private String covers;
External(String modelName, double baseArea, Shape shape, String covers) {
   super(modelName, baseArea, shape);
   this.covers = covers;
}
```

Имплементира интерфејс Cover и његову методу дефинише тако да омогућава испис поруке о покривености базена у зависности од вредности атрибута cover.

```
//MESSAGE ABOUT AVAILABILITY OF COVERS
@Override
public void cover() {
   if(covers.equals("Yes")){System.out.println("Pool cover is available");}
   else {System.out.println("Pool cover is not available");}
}
```

Редефинише наслеђен метод showObjectData() тако што већ дефинисаном испису додаје редефинисану поруку из методе cover().

```
 //SHOWING COMMON PROPERTIES FROM POOL, INCLUDING EXTERNAL POOL COVERS
   @Override
   public void showObjectData() {
      super.showObjectData();
      System.out.print(" -covers: ");
      cover();

   }
}
```

3.6.3. Класа Internal

Internal класа је поткласа класе Pool и поред наслеђених атрибута садржи свој pumpNumber који даје информацију о броју пумпи. Конструктору се поред наслеђених атрибута прослеђује и приватан. Класа садржи getter који враћа вредност атрибута pumpNumber. Редефинише методу showObjectData() и испису додаје вредност свог атрибута.

```
        public class Internal extends Pool {
  //NUMBER OF PUMPS FOR INTERNAL POOL
  private int pumpNumber;

  Internal(String modelName, double baseArea, Shape shape, int pumpNumber) {
    super(modelName, baseArea, shape);
    this.pumpNumber = pumpNumber;
  }

  //GETTER FOR PUMP NUMBERS
  private int getPumpNumber() {
    return pumpNumber;
  }

  //SHOWING COMMON PROPERTIES FROM POOL, INCLUDING INTERNAL PUMP NUMBERS
  @Override
  public void showObjectData() {
    super.showObjectData();
    System.out.print(" -pump number: " + getPumpNumber());
    System.out.println();
  }
}
```

3.6.4.Класа Helper

Класа Helper је апстрактна класа. Апстракција је процес сакривања одређених детаља и приказивања само функционалности кориснику. Апстрактна класа је ограничена класа која се не може користити за иницијализацију објекта. Може имати апстрактне и регуларне методе.

Класа Helper садржи статичку методу showSet која као аргумент прима направљен сет базена и исписује га. Сетови су неуређене колекције објеката. Static је кључна реч и код статичког метода нема потребе за креирањем објекта који би га позвао. Void је тип који враћа метод, не враћа никакву вредност.

```
        //HELPER CLASS
 abstract class Helper {

   //SHOWING SET OF POOLS
static void showSet(Set<Pool> pools){
   for(Pool pool :pools){
      pool.showObjectData();
   }
}
```

Метода makeSortedList као аргумент прима прослеђен сет базена и чува га у листи. Затим их пореди помоћу дефинисаног компаратора по површини основе и враћа тако сортирану листу.

```
   //MAKING SORTED LIST OF POOLS FROM SET
static ArrayList<Pool> makeSortedList(Set<Pool> pools){
   ArrayList<Pool> poolsList=new ArrayList<>(pools);
   poolsList.sort(new CompareBaseArea());
   return poolsList;

}
```

Метода makeInternalList као аргумент прима сет базена и дефинише листу. Сваком објекту из сета проверава тип и ако је тип Internal додаје га у своју дефинисану листу и затим је враћа. Исти принцип је примењен на методу makeExternalList. Једина разлика је у типу листе. ArrayList представља проширив низ. За чување референци на елементе интерно користи низ. LinkedList је имплементација List интерфејса и елементи се складиште у виду повезане листе.

```
    //MAKING LIST OF INTERNAL POOLS FROM SET
static ArrayList<Pool> makeInternalList(Set<Pool> pools){
   ArrayList<Pool> internalpoolsList=new ArrayList<>();
   for(Pool pool: pools ){
      if(pool.getClass()==Internal.class){
         internalpoolsList.add(pool);
      }
   }
   return internalpoolsList;
}
          //MAKING LIST OF EXTERNAL POOLS FROM SET
static LinkedList<Pool> showExternalList(Set<Pool> pools){
   LinkedList<Pool> externalpoolsList=new LinkedList<>();
   for(Pool pool: pools ){
      if(pool.getClass()==External.class){
         externalpoolsList.add(pool);
      }
   }
   return externalpoolsList;
}
```

Метода showList као аргумент прима листу базена коју исписује.

```
          //SHOWING LIST OF POOLS
static void showList(List<Pool> poolsList){
   for(Pool pool:poolsList ){
      pool.showObjectData();
   }
}
```

3.6.5. Класа WriteInFile

Класа WriteInFile служи за упис прослеђене листе базена у текстуалну датотеку pool.txt. Имплементира Runnable интерфејс, који треба бити имплементиран од стране било које класе чије ће инстанце бити извршене нитима. Класа мора да дефинише методу без аргумената која се зове run() .

Дефинисана је листа за упис у фајл као и њен конструктор и get метода. Нит представља једну секвенцу наредби које се извршавају унутар програма. Извршавање нити може бити истовремено, при чему свака нит извршава различите задатке. Та особина у Јави се назива multithreading и омогућава да се један програм подели у мање целине које могу да се извршавају истовремено што смањује укупну брзину програма.

```
        public class WriteInFile implements Runnable {
//FORWARDING THE LIST FOR WRITING IN FILE
private List<Pool> poolsList;
public WriteInFile(List<Pool> poolsList) {
   this.poolsList = poolsList;
}
public List<Pool> getPoolsList() {
   return poolsList;
}
```

Нит представља једну секвенцу наредби које се извршавају унутар програма. Извршавање нити може бити истовремено, при чему свака нит извршава различите задатке. Та особина у Јави се назива multithreading и омогућава да се један програм подели у мање целине које могу да се извршавају истовремено што смањује укупну брзину програма.

```
//RUN THE THREAD
@Override
public  void  run() {
try (FileWriter writer = new FileWriter("pool.txt", true)) {
   System.out.println("Writing in file is active: " + Thread.currentThread().isAlive());
   for (Pool pool : getPoolsList()) {
      //WRITING MODEL NAME AND TYPE IN POOL.TXT
         writer.write(pool.getModelName() + "[ " + pool.getObjectType() + " ]" + System.lineSeparator());
         //CURRENT THREAD PAUSES SO OTHERS CAN EXECUTE
      Thread.yield();
   }
} catch (IOException e) {
   e.printStackTrace();
 } } }
```

Try блок омогућава дефинисање блока кода који ће бити тестиран на грешке приликом извршавања. Као ресурс му је предат објекат типа FileWriter који служи за упис података, у карактерима, у фајл. Подаци ће се уписивати у текстуалну датотеку pool.txt . Други параметар append је постављен на true што значи да ће подаци бити уписивани на крај фајла.

Thread.currentThread().isAlive() ће вратити вредности true по извршавању тренутне нити. Thread.yield() је статички метод који зауставља тренутну нит и даје шансу за извршавање осталим нитима које чекају истог приоритета.

Catch омогућава дефинисање блока кода који ће се извршити ако се грешка појави у try блоку. Хвата IOException који се појављује када дође до грешке у улазно-излазној операцији и помоћу e.printStackTrace() методе исписује грешку са осталим детаљима у виду броја реда и имена класе где се грешка појавила.

3.6.6. Класа ReadFromFile

Класа ReadFromFile служи за читање уписаних листа базена из датотеке pool.txt и њихов испис. Као и класа WriteInFile имплементира Јавин интерфејс Runnable. У try блоку као ресурс му је предат објекат типа FileReader и наведен је фајл из ког ће се вршити читање података. FileReader је класа из java.io пакета која се користи за читање података из фајла. Враћа податке у виду бајтова.

```
        @Override
public void run() {
    try (FileReader reader = new FileReader("pool.txt")) {

        int i;
        try {
            //THREAD PAUSE THE EXECUTION FOR 1000ms
            Thread.sleep(1000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        System.out.println("Reading from file is active: "+Thread.currentThread().isAlive());
        //READING TILL END OF FILE(-1)
        while ((i = reader.read()) != -1) {
            //BYTE FORMAT TO CHAR
            System.out.print((char) i);
        }

    } catch (IOException e) {
        e.printStackTrace(); } }}
```

У другом try блоку коришћена је Thread.sleep() метода уз помоћ које се извршавање тренутне нити суспендује за наведени временски период. У овом случају тај временски период је 1000 милисекунди. На тај начин се омогућава да процесорско време буде доступно другим нитима. InterruptedException изузетак се избацује када је нит прекинута док су остале у стању wait, sleep или окупиране. Спада у проверљиве изузетке.

Метода read() служи за читање једног карактера. Враћа карактер или -1 ако је достигао крај фајла. Услов у while петљи означава да ће reader читати карактер по карактер све док не стигне до краја фајла и приликом исписа ће вршити конверзију у char тип. Хватање изузетка је исто као у класи WriteInFile.

3.6.7. Класа Main

Применом полиморфизма направљена су по три објекта типа Internal и External и као аргументе, који су дефинисани конструктором у класи Pool, прослеђени су им жељени атрибути.

```
        //MAKING 3 OBJECTS OF INTERNAL TYPE
Pool internalPool1=new Internal("AquaCave Swimming Pool",111.5,Shape.RECTANGLUAR,10);
Pool internalPool2=new Internal("Blue Fish",400,Shape.CIRCULAR,12);
Pool internalPool3=new Internal("Shellfish",110.3,Shape.CUSTOM,8);

    //MAKING 3 OBJECTS OF EXTERNAL TYPE
Pool externalPool1=new External("Laguna",300.0,Shape.RECTANGLUAR,"Yes");
Pool externalPool2=new External("Sunny",173.3,Shape.RECTANGLUAR,"No");
Pool externalPool3=new External("Lux",200.9,Shape.CUSTOM,"No");
```

Направљен је нови сет pools и сачуван је као HashSet, додати су му направљени објекти и исписан је помоћу методе из Helper класе showSet() којој је као аргумент прослеђен направљен HashSet базена. HashSet се користи за креирање колекција које користе хеш табелу за складиштење. Не прати ред по ком се објекти уносе већ се објекти уносе по њиховом хеш коду.

```
        //MAKING,ADDING,SHOWING HASHSET OF POOLS
Set<Pool> pools=new HashSet<>();
pools.add(internalPool1);
pools.add(internalPool2);
pools.add(internalPool3);
pools.add(externalPool1);
pools.add(externalPool2);
pools.add(externalPool3);
System.out.println("-----------------------------------SET OF POOLS----------------------------------------");
    Helper.showSet(pools);
    System.out.println("---------------------------------------------------------------------------------------");
```

Помоћу методе makeSortedList() из класе Helper, којој је као аргумент предат направљен сет базена, направљена је листа базена сортирана по површини основе .

Помоћу методе makeInternalList() из класе Helper, којој је као аргумент предат направљен сет базена, направљена је листа базена која се састоји само од објеката типа Internal.

Помоћу методе makeExternalList() из класе Helper, којој је као аргумент предат направљен сет базена, направљена је листа базена која се састоји само од објеката типа External.

Све листе су исписане у конзоли помоћу методе showList() из класе Helper којој се као аргумент прослеђује одговарајућа листа за испис.

```
        //SHOWING LIST OF POOLS SORTED BY BASE AREA
System.out.println("----------------------------------POOLS BY BASE AREA-------------------------------------");
    List<Pool> sortedList = Helper.makeSortedList(pools);
    Helper.showList(sortedList);
System.out.println("----------------------------------------------------------------------------------------");

//SHOWING LIST OF INTERNAL POOLS / ARRAY LIST
System.out.println("----------------------------------INTERNAL POOLS-----------------------------------------");
     List<Pool> internalList=Helper.makeInternalList(pools);
     Helper.showList(internalList);
System.out.println("----------------------------------------------------------------------------------------");

//SHOWING LIST OF EXTERNAL POOLS / LINKED LIST
System.out.println("----------------------------------EXTERNAL POOLS-----------------------------------------");
List<Pool> externalList=Helper.showExternalList(pools);
     Helper.showList(externalList);
System.out.println("----------------------------------------------------------------------------------------");
```

Направљен је објекат типа Thread, нит read, за већ направљен задатак ReadFromFile који служи за читање података из текстуалне датотеке.

Направљен је објекат типа Thread, нит writeInternal, за већ направљен задатак WriteInFile, коме се као аргумент прослеђује већ направљена листа internalList, и служи за упис прослеђене листе у текстуалну датотеку. Исти принцип је примењен за нит writeExternal.

Нити за упис у датотеку су покренуте start() методом која служи да започне извршавање нити. Резултат те методе је да се две нити извршавају конкурентно, тренутна позвана start() методом и друга нит која извршава своју run() методу. Метода start() интерно позива run() методу Јавиног интерфејса Runnable да изврши код из run() методе у посебној нити.

```
Thread read=new Thread(new ReadFromFile());
Thread writeInternal=new Thread(new WriteInFile(internalList));
Thread writeExternal=new Thread(new WriteInFile(externalList));
//STARTING THE THREADS
writeInternal.start();
writeExternal.start();
//ACTIVE LOOP WHILE THREADS ARE ACTIVE
while (writeExternal.isAlive() && writeInternal.isAlive()){
    System.out.print("");
}
//READING FROM FILE
read.start();
```

Петља while је петља која ће бити активна све док су нити за упис активне. Метода isAlive() даје информацију о тренутном стању петље и даће вредност true ако је петља тренутно активна.

Покренута је нит read методом start() и врши читање из датотеке pool.txt и испис у конзоли.

3.6.8. Испис у конзоли

```
-----------------------------------SET OF POOLS-----------------------------------------
[MJ]- type: Internal - name: Shellfish -base area: 110.3 -shape: CUSTOM  -pump number: 8
[MJ]- type: External - name: Sunny -base area: 173.3 -shape: RECTANGLUAR  -covers: Pool cover is not available
[MJ]- type: Internal - name: Blue Fish -base area: 400.0 -shape: CIRCULAR  -pump number: 12
[MJ]- type: Internal - name: AquaCave Swimming Pool -base area: 111.5 -shape: RECTANGLUAR  -pump number: 10
[MJ]- type: External - name: Laguna -base area: 300.0 -shape: RECTANGLUAR  -covers: Pool cover is available
[MJ]- type: External - name: Lux -base area: 200.9 -shape: CUSTOM  -covers: Pool cover is not available
----------------------------------------------------------------------------------------
-----------------------------------POOLS BY BASE AREA-----------------------------------
[MJ]- type: Internal - name: Shellfish -base area: 110.3 -shape: CUSTOM  -pump number: 8
[MJ]- type: Internal - name: AquaCave Swimming Pool -base area: 111.5 -shape: RECTANGLUAR  -pump number: 10
[MJ]- type: External - name: Sunny -base area: 173.3 -shape: RECTANGLUAR  -covers: Pool cover is not available
[MJ]- type: External - name: Lux -base area: 200.9 -shape: CUSTOM  -covers: Pool cover is not available
[MJ]- type: External - name: Laguna -base area: 300.0 -shape: RECTANGLUAR  -covers: Pool cover is available
[MJ]- type: Internal - name: Blue Fish -base area: 400.0 -shape: CIRCULAR  -pump number: 12
----------------------------------------------------------------------------------------
-----------------------------------INTERNAL POOLS---------------------------------------
[MJ]- type: Internal - name: Shellfish -base area: 110.3 -shape: CUSTOM  -pump number: 8
[MJ]- type: Internal - name: Blue Fish -base area: 400.0 -shape: CIRCULAR  -pump number: 12
[MJ]- type: Internal - name: AquaCave Swimming Pool -base area: 111.5 -shape: RECTANGLUAR  -pump number: 10
----------------------------------------------------------------------------------------
-----------------------------------EXTERNAL POOLS---------------------------------------
[MJ]- type: External - name: Sunny -base area: 173.3 -shape: RECTANGLUAR  -covers: Pool cover is not available
[MJ]- type: External - name: Laguna -base area: 300.0 -shape: RECTANGLUAR  -covers: Pool cover is available
[MJ]- type: External - name: Lux -base area: 200.9 -shape: CUSTOM  -covers: Pool cover is not available
----------------------------------------------------------------------------------------
Writing in file is active: true
Writing in file is active: true
Reading from file is active: true


 Shellfish[ Internal ]
 Blue Fish[ Internal ]
 AquaCave Swimming Pool[ Internal ]
 Sunny[ External ]
 Laguna[ External ]
 Lux[ External ]
```

4. Поставка пројектног задатка

Направити конзолну апликацију за потребе грађевинске фирме. Сходно потребама клијента, апликација треба да испуни следеће захтеве. Треба обезбедити могућност рада са објектима типа Internal и External. Заједничка својства ових објеката јесу: константа која треба да садржи податак о типу објекта, променљива која се односи на име модела, променљива која се односи на површину основе, променљива која се односи на облик (облик може бити искључиво једна од следеће три вредности: RECTANGULAR, CIRCULAR, CUSTOM).

За разлику од објеката типа External, објекти типа Internal имају и својство које садржи у себи податак о броју пумпи. За прављење ових објеката се користи конструктор који треба да додели вредност свим њиховим променљивама. Заједничка понашања ових објеката су могућност читања (не и постављања) вредности свих заједничких својстава, могућност приказа података о објекту (имплементирано помоћу функционалног интерфејса) и то: порука треба да почиње иницијалима (у угластим заградама) студента који ради пројекат (у супротном треба избацити изузетак који припада категорији непроверљивих изузетака са адекватном поруком), након иницијала треба исписати сва својства објекта за који се метод позива, порука не сме садржати карактере „{“, „}“ и „=“ (у супротном треба избацити изузетак који припада категорији непроверљивих изузетака и то са поруком: "You tried to use "{", "}" and/or "=" in presentation message!", обратити пажњу на знаке навода код навођења забрањених карактера у поруци), наведене изузекте направити и применити тако да одговарају захтевима(пример: [NS] – type: Internal – name: Orca – base area: 45 – shape: RECTANGULAR – pump no: 3), могућност међусобног поређења свих објеката (без обзира на конкретни тип ком припадају) на основу заједничког својства који се односи на површину основе. За разлику од објеката типа Internal, објекти типа External имају и понашање које се тиче приказа поруке о могућности наткривања базена (такође имплементирано помоћу функционалног интерфејса).

Обезбедити начин за приказ података свих објеката који се налазе у генеричкој колекцији независно од њеног конкретног типа и типа предметних објеката (Internal и External) који се налазе у њој, начин за добијање (нове) колекције типа ArrayList сачињене од међусобно сортираних објеката из предате генеричке колекције независно од њеног конкретног типа и типа предметних објеката који се налазе у њој, начин за добијање колекције типа ArrayList сачињене искључиво од објеката типа Internal, а на основу предате генеричке колекције независно од њеног конкретног типа и типа предметних објеката који се налазе у њој, начин за добијање колекције типа LinkedList сачињене искључиво од објеката типа External, а на основу предате генеричке колекције независно од њеног конкретног типа и типа предметних објеката који се налазе у њој.

НАПОМЕНА: За потребе дефинисања свих помоћних података и функционалности треба обезбедити једну помоћну класу која се не може инстанцирати и којој се не може мењати унутрашње стање (сви елементи треба да јој буду непроменљиви).

Направити класу задатак за упис у фајл и то: приликом прављења инстанце задатка, у виду аргумента, омогућити предају генеричке листе која може примити објекте оба предметна типа, приликом извршења задатка обезбедити упис података у локалну текстуалну датотеку „pool.txt“, у датотеку уписивати следеће податке: име модела и тип објекта који за који се врши упис у угластој загради , сваки појединачни запис треба да буде у новом реду (пример: Orca [Internal]), након уписаног сваког појединачног записа извршити заустављање тренутне нити извршења до момента док се остале са којима је она у вези не изврше.

Направити класу задатак за читање из фајла и то: приликом извршења задатка треба обезбедити читање свог садржаја фајла, непосредно пред отпочињање процеса читања стопирати тренутну нит за 1000 ms.

У главној класи уз употребу полиморфизма, направити по три објекта оба предметна типа, направљене објекте сместити у генерички сет који може да прими оба типа предметних објекта, приказати све објекте претходно направљеног и попуњеног сета (употребом обезбеђеног начина), направити листу сортираних објектата од објеката претходно приказаног сета (употребом обезбеђеног начина), приказати објекте из сортиране листе (употребом обезбеђеног начина),од објеката сета направити одговарајући тип листе објеката типа Internal (употребом обезбеђеног начина), од објеката сета направити одговарајући тип листе објеката типа External (употребом обезбеђеног начина), направити нит задатка којим се врши упис у фајл (предати му новонаправљену листу Internal објеката), направити нит задатка којим се врши упис у фајл (предати му новонаправљену листу External објеката), направити нит задатка читања из фајла, покренути нити уписа у фајл, дефинисати петљу која ће се бити активна докле год су активне и нити уписа у фајл, покренути нит читања из фајла.

НАПОМЕНА: Уредити структуру пројекта и очистити код од непотребних елемената.

5. Закључак

Предност објектно оријентисаног програмирања је та што омогућава решавање проблема међусобном сарадњом објеката унутар програма и пројекат се на тај начин разлаже на мање логичке целине. Програмерима је омогућено креирање модула који не морају да се мењају када се дода нова врста објекта. Креира се нови објекат који наслеђује особине од постојећег објекта и програми се лако мењају.

6. Литература

[1] www.javatpoint.com

[2] www.baeldung.com

[3] www.stackoverflow.com

[4] www.geeksforgeeks.org

[5] docs.oracle.com

[6] www.w3schools.com

[7] www.it-akademija.com