**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ “НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ”**

Лабораторная работа №3

по дисциплине «Технологии и методы программирования»

Классы-коллекции.

Группа: **АВТ-042**

Студенты: **Антонов А.Р.**

Преподаватель: **Булатов А.Д**

Новосибирск 2022

1. **Цель**

Изучить особенности реализации классов-коллекций в Kotlin.

1. **Задание**

***Вариант 10***

Коллекция для хранения объектов: Vector

Коллекция для хранения и поиска уникальных идентификаторов: TreeSet

Коллекция для хранения времени рождения объектов: HashMap

Доработать программу, созданную в лабораторной работе № 2:

1. добавить генерируемым объектам понятия «время рождения» и «время жизни». Время рождения устанавливается в момент генерации объекта, и по значению соответствует времени, прошедшему от начала симуляции. Время жизни – время, через которое объект должен исчезнуть, считая от времени рождения;

2. вынести установку параметров времени жизни объектов в пользовательский интерфейс. Для каждого типа объекта должно задаваться собственное время. Рекомендуется использовать текстовые поля, но следуют помнить о проверке на ввод некорректных данных;

3. добавить генерируемым объектам уникальные целочисленные идентификаторы (случайные числа), которые назначаются при генерации объекта. Для хранения сгенерированных идентификаторов используйте коллекцию удобную для поиска по варианту;

4. использовать коллекции по варианту. При генерации объекта происходит добавление его в коллекцию (в класс добавить поле идентификатора), а во вторую коллекцию: идентификаторы существующих объектов, в третью идентификатор ̶ время рождения. При возникновении события по таймеру обойдите коллекцию и удалите все объекты, время жизни которых истекло, а также все данные во вспомогательных коллекциях.

5. добавьте в панель управления кпопку «Текущие объекты». По нажатию на эту кнопку появляется модальное диалоговое окно, содержащее список всех «живых» объектов на момент нажатия со временем их рождения (время рождения – ключ). В класс диалогового окна должна передаваться коллекция с хранением объектов по времени рождения.

1. **UML-диаграмма**

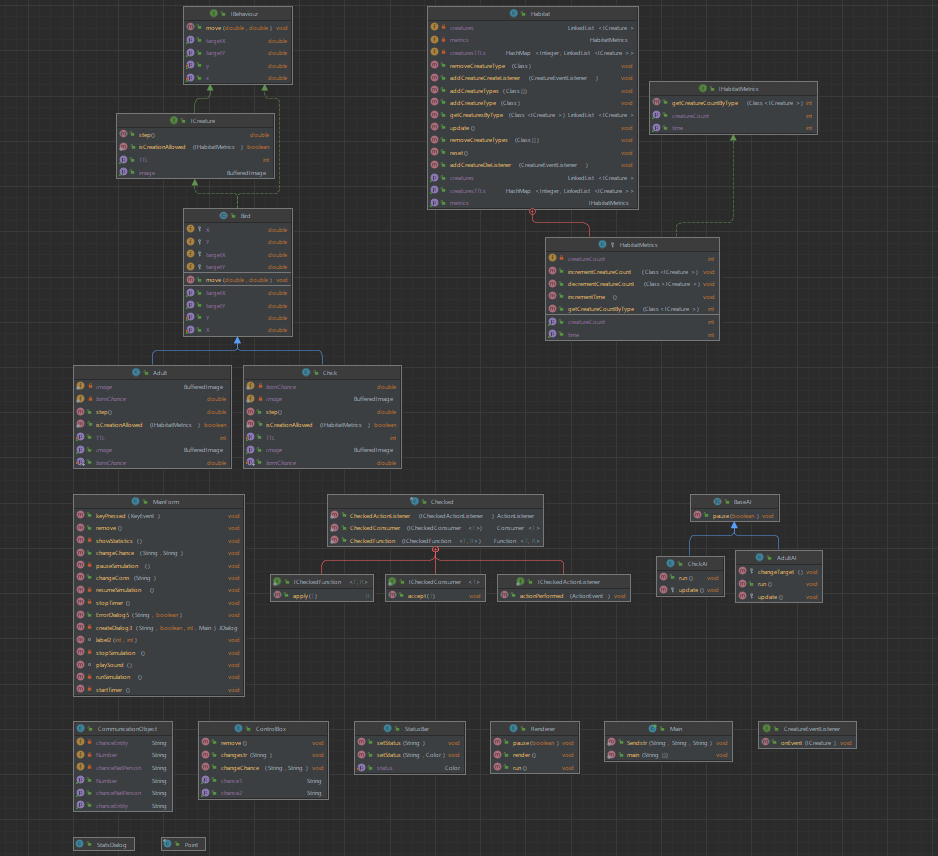


Рисунок 1 UML-диаграмма

1. **Пример работы программы**

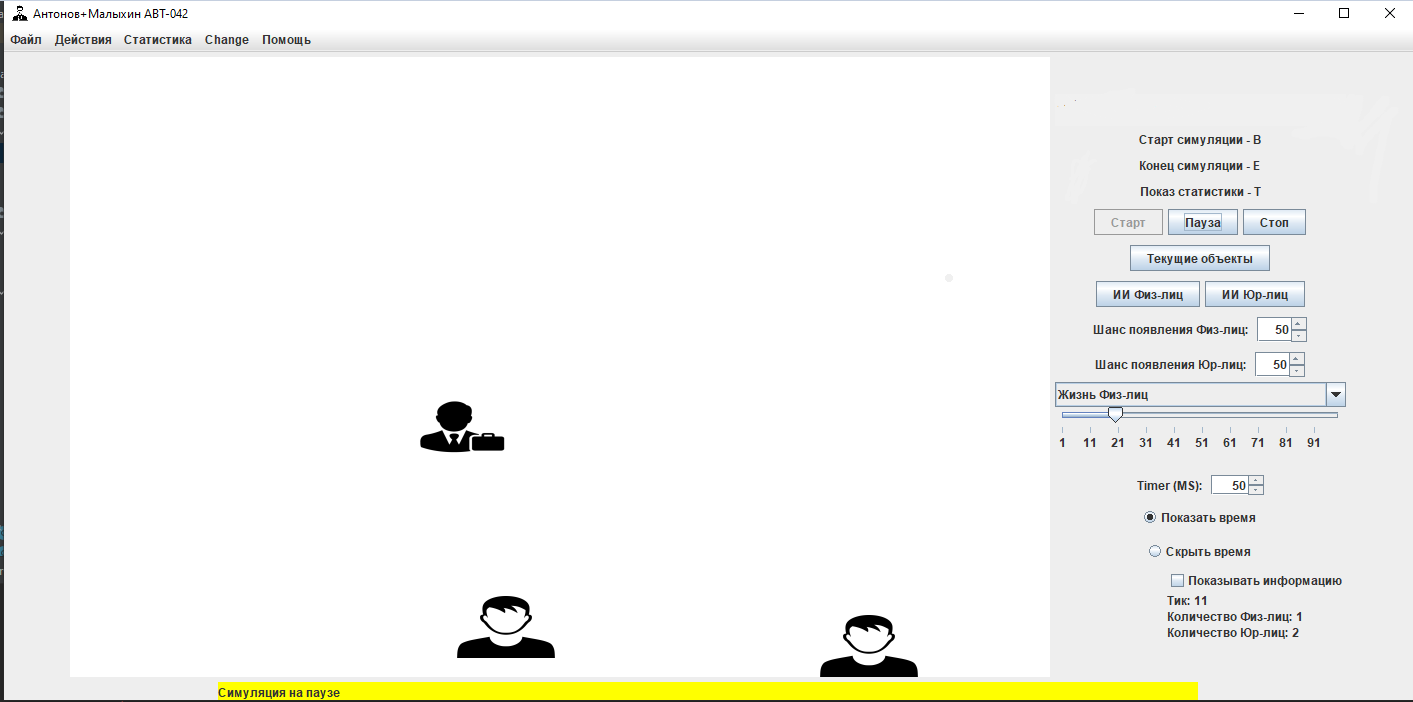


Рисунок 2 Пример работы программы

1. **Вывод**

Мы изучили особенности реализации классов-коллекций в Kotlin. Доработали программу, созданную в лабораторной работе № 2. Были выполнены все требования согласно варианту.

**Приложение**

**Class Habitat**

package lab.habitat;  
  
import java.util.\*;  
  
  
public class Habitat {  
 private LinkedList<CreatureEventListener> creatureCreateListeners = new LinkedList<>();  
 private LinkedList<CreatureEventListener> creatureDieListeners = new LinkedList<>();  
 private LinkedList<Class<? extends ICreature>> creatureTypes = new LinkedList<>();  
 private HabitatMetrics metrics = new HabitatMetrics();  
  
 private int ticks = 0;  
 private LinkedList<ICreature> creatures = new LinkedList<>();  
 private HashMap<Integer, LinkedList<ICreature>> creaturesTTLs = new HashMap();  
  
  
 public void update() {  
 ticks++;  
 metrics.incrementTime();  
 // ограничение рождаемости  
 if(creatures.size() >= 20)  
 return;  
  
 // create new creatures  
 creatureTypes.forEach(T -> {  
 try {  
 if((Boolean)T.getMethod("isCreationAllowed", IHabitatMetrics.class).invoke(null, metrics)) {  
 // *TODO: search for params constructor?* ICreature c = T.getConstructor().newInstance();  
 c.setX(Math.*random*());  
 c.setY(Math.*random*());  
 creatures.add(c);  
  
 LinkedList<ICreature> cs = creaturesTTLs.getOrDefault(ticks + c.getTTL(), new LinkedList<>());  
 cs.add(c);  
 creaturesTTLs.put(ticks + c.getTTL(), cs);  
  
 metrics.incrementCreatureCount(T);  
 creatureCreateListeners.forEach(e -> e.onEvent(c));  
 }  
 }  
 catch(Exception ex) {  
 ex.printStackTrace();  
 }  
 });  
  
 // and kill too old  
 LinkedList<ICreature> cs = creaturesTTLs.getOrDefault(ticks, null);  
 if(cs != null) {  
 cs.forEach(c -> {  
 creatures.removeFirstOccurrence(c);  
  
 metrics.decrementCreatureCount(c.getClass());  
 creatureDieListeners.forEach(e -> e.onEvent(c));  
 });  
  
 creaturesTTLs.remove(ticks);  
 }  
 }  
  
 public void reset() {  
 ticks = 0;  
 metrics = new HabitatMetrics();  
 creatures.clear();  
 creaturesTTLs.clear();  
 }  
  
  
 public IHabitatMetrics getMetrics() {  
 return metrics;  
 }  
  
 public void addCreatureType(Class creatureType) throws IllegalArgumentException {  
 if(!ICreature.class.isAssignableFrom(creatureType)) {  
 throw new IllegalArgumentException(creatureType.getName() + " doesn't implement Creature interface!");  
 }  
  
 if(creatureTypes.contains(creatureType)) {  
 throw new IllegalArgumentException(creatureType.getName() + " already added to current habitat");  
 }  
  
 // but this assignment IS checked!  
 creatureTypes.add(creatureType);  
 }  
  
 public LinkedList<ICreature> getCreaturesByType(Class<? extends ICreature> type) {  
 LinkedList<ICreature> creaturesByType = new LinkedList<>();  
  
 for (ICreature c : creatures) {  
 if(c.getClass() == type)  
 creaturesByType.add(c);  
 }  
  
 return creaturesByType;  
 }  
  
 public LinkedList<ICreature> getCreatures() {  
 return (LinkedList<ICreature>) creatures.clone();  
 }  
  
 public HashMap<Integer, LinkedList<ICreature>> getCreaturesTTLs() {  
 return (HashMap<Integer, LinkedList<ICreature>>) creaturesTTLs.clone();  
 }  
  
  
 public void addCreatureTypes(Class ... creatureTypes) throws IllegalArgumentException {  
 for (Class creatureType : creatureTypes) {  
 addCreatureType(creatureType);  
 }  
 }  
  
 public void removeCreatureType(Class creatureType) {  
 creatureTypes.remove(creatureType);  
 }  
  
 public void removeCreatureTypes(Class ... creatureTypes) throws IllegalArgumentException {  
 for (Class type : creatureTypes) {  
 removeCreatureType(type);  
 }  
 }  
  
  
 public void addCreatureCreateListener(CreatureEventListener e) {  
 creatureCreateListeners.add(e);  
 }  
  
 public void addCreatureDieListener(CreatureEventListener e) {  
 creatureDieListeners.add(e);  
 }  
  
  
 protected class HabitatMetrics implements IHabitatMetrics {  
 private int creatureCount = 0;  
 private int ticks = 0;  
 private HashMap<Class<? extends ICreature>, Integer> creatureCountByType = new HashMap<>();  
  
  
 public int getCreatureCount() {  
 return creatureCount;  
 }  
  
 public int getCreatureCountByType(Class<? extends ICreature> type) {  
 return creatureCountByType.getOrDefault(type, 0);  
 }  
  
 public int getTime() {  
 return ticks;  
 }  
  
  
 public void incrementCreatureCount(Class<? extends ICreature> type) {  
 ++creatureCount;  
  
 int count = creatureCountByType.getOrDefault(type, 0);  
 creatureCountByType.put(type, ++count);  
 }  
  
 public void decrementCreatureCount(Class<? extends ICreature> type) {  
 --creatureCount;  
  
 int count = creatureCountByType.get(type);  
 creatureCountByType.put(type, --count);  
 }  
  
 public void incrementTime() {  
 ++ticks;  
 }  
 }  
}