Министерство науки и высшего образования Российской Федерации

Новосибирский государственный технический университет

Лабораторная работа №3 по дисциплине  
«Технология программирования» на тему

***Классы-коллекции***

Факультет: АВТФ Преподаватель:

Группа: АВТ-810 Михайленко Д. А.

Студенты: Степаненко Владислав

Новосибирск

2020

#### Практические задания

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; Для обеспечения доступа к коллекциям используйте структурный паттерн Singleton.
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. Типы коллекций задаются вариантом.

Вариант 7

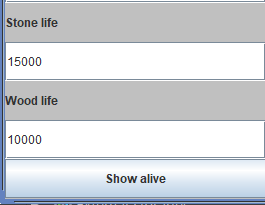
1.Коллекция для хранения объектов: Vector

2.Коллекция для хранения и поиска уникальных идентификаторов: HashSet

3.Коллекция для хранения времени рождения объектов: TreeMap

**Ход работы:**

Объектам добавлены параметры времени жизни и времени появления, в пользовательском интерфейсе добавлено соответствующее поле ввода.

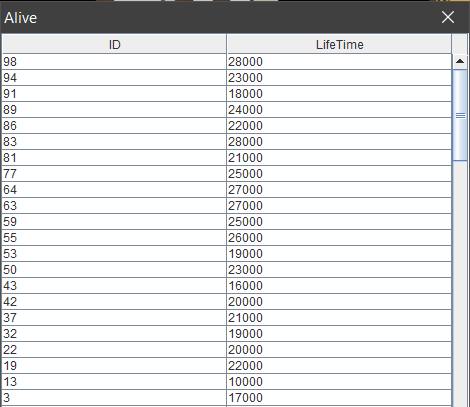


Каждому объекту присваивается случайный id, id не повторяются.

В классе Singleton созданы 3 типа коллекций, Vector, HashSet,TreeMap

Они соответственно хранят объекты, id, id/время появления.

Добавлена кнопка по нажатию которой выводится информация о “живых” объектах симуляции



**Листинг программы(основные изменения):**

public void update (float time, JPanel simulation\_panel)  
{  
 if (time/1000 % t1 == 0 && time!=0 )  
 {  
 if (Math.*random*() < p1)  
 {  
 int id = houseSingleton.idRandom();  
 House stoneHouse = allFactory.createStone((int)(Math.*random*()\*simulation\_panel.getWidth()),(int)(Math.*random*()\*simulation\_panel.getHeight()));  
 stoneHouse.setAppTime((int)time);  
 stoneHouse.setLifePeriod(stoneLifeTime);  
 stoneHouse.setId(id);  
 houseSingleton.houseList.add(houseSingleton.houseList.size(),stoneHouse);  
 houseSingleton.mapList.put(id,(long)stoneHouse.appTime);  
 houseSingleton.idList.add(id);  
 counter++;  
 }  
 }  
 if (time/1000 % t2 == 0 && time!=0 )  
 {  
 if (Math.*random*() < p2) {  
 int id = houseSingleton.idRandom();  
 House woodHouse = allFactory.createWood((int) (Math.*random*() \* simulation\_panel.getWidth()), (int) (Math.*random*() \* simulation\_panel.getHeight()));  
 woodHouse.setAppTime((int)time);  
 woodHouse.setLifePeriod(stoneLifeTime);  
 woodHouse.setId(id);  
 houseSingleton.houseList.add(houseSingleton.houseList.size(),woodHouse);  
 houseSingleton.mapList.put(id,(long)woodHouse.appTime);  
 houseSingleton.idList.add(id);  
 counter++;  
 }  
 }  
 for (int i=0;i< houseSingleton.houseList.size();i++)  
 {  
 if (houseSingleton.houseList.get(i).appTime + houseSingleton.houseList.get(i).lifePeriod <= time)  
 {  
 houseSingleton.mapList.remove(houseSingleton.houseList.get(i).id);  
 houseSingleton.idList.remove(houseSingleton.houseList.get(i).id);  
 houseSingleton.houseList.remove(i);  
 counter--;  
 }  
 }  
 int w = simulation\_panel.getWidth();  
 int h = simulation\_panel.getHeight();  
 BufferedImage bufferedImage = new BufferedImage(w,h,BufferedImage.*TYPE\_INT\_RGB*);  
 for (int i=0;i< houseSingleton.houseList.size();i++)  
 {  
 bufferedImage.getGraphics().drawImage(houseSingleton.houseList.get(i).getImage(),  
 houseSingleton.houseList.get(i).getX(), houseSingleton.houseList.get(i).getY(), null);  
 }  
 simulation\_panel.getGraphics().drawImage(bufferedImage,0,0,w,h,null);  
  
  
  
}

public class Singleton  
{  
 private Singleton(){}  
  
 private static Singleton *instance*;  
 public Vector<House> houseList = new Vector<>(1000);  
 public HashSet<Integer> idList = new HashSet<>(1000);  
 public TreeMap<Integer, Long> mapList = new TreeMap<>();  
 public static Singleton getInstance()  
 {  
 if(*instance* == null)  
 *instance* = new Singleton();  
 return *instance*;  
 }  
 public int idRandom()  
 {  
 int id;  
 do {  
 id = (int)(Math.*random*()\*100);  
 } while (idList.contains(id));  
 return id;  
  
 }  
 public void clear()  
 {  
 houseList.clear();  
 idList.clear();  
 mapList.clear();  
 }  
  
}

public void showAlive(UserInterface ui)  
{  
 JDialog dialog = new JDialog();  
 dialog.setTitle("Alive");  
 String[] cols = {"ID", "LifeTime"};  
 int rows = 100;  
 JTable table = new JTable(new DefaultTableModel(cols, rows));  
 TableModel tableModel = table.getModel();  
  
 int i = 0;  
  
 for (Iterator it = houseSingleton.mapList.descendingMap().entrySet().iterator(); it.hasNext(); i++)  
 {  
 Map.Entry<Integer, String> entry = (Map.Entry<Integer, String>) it.next();  
 tableModel.setValueAt(entry.getValue(), i, 1);  
 tableModel.setValueAt(entry.getKey(), i, 0);  
 }  
 JScrollPane scrollPane = new JScrollPane(table);  
 dialog.add(scrollPane);  
 dialog.setSize(new Dimension(250, 100));  
 dialog.setResizable(false);  
 dialog.setVisible(true);  
 dialog.pack();  
}