**Лабораторная работа №5**

**Задание 1.**

Реализовать простейший текстовый квест, в котором должны быть следующие функции:

* Герой путешествует по локациям, которые идут друг за другом;
* На каждой локации происходит диалог с персонажем, который даёт игроку загадку. За правильный ответ игрок получает очки, за неправильный - отнимается здоровье; Цель игры - пройти десять локаций и не умереть.
* Все локации имеют у себя в классе счётчик сложности;
* Локации группируются между собой случайным образом, но в рамках своей сложности (то есть, игрок не должен первой локацией попасть на самый сложный уровень); Такую сортировку и фильтрацию можно сделать при помощи методов StreamAPI.
* Для ясности, условное расположение локаций в массиве в виде чисел сложности заданий:
* До сортировки [1, 1, 2, 3, 5, 6, 1, 5, 6, 4, 2, 1, 3]
* После: [1, 1, 1, 1, 2, 2, 3, 3, 4, 5, 6, 6] - отсюда выбирается по одной-две локации для каждой сложности и итоговый набор получается из 10 локаций с постепенным увеличением сложности.
* Вывод текста происходит в любую консоль (IDE или Терминал);
* В игре должна присутствовать возможность сохранения текущего состояния - локации, параметров игрока - в файл для последующей загрузки при следующем запуске;
* Желательно, чтобы текст не был однотонным (но в меру! “Вырвиглазное” оформление считается за невыполненную работу);
* NPC обращаются к игроку по имени, введенному в начале игры;

**Набор исходных данных:**

Программа использует следующие исходные данные:

Имена для NPC:

String[] nameNPC = {"Обычный Лис", "Пушистый Енот", "Красный Лис", "Пустынный Лис",  
 "Лесной Лис", "Загодочный Лис", "Игривый Енот", "Зеленый Лис", "Волшеный Лис", "Редкий Лис",  
 "Разбойник-Енот", "Вор-Енот", "Дружелюбный Енот", "Нахальный Енот", "Очаровательный Енот"};  
Описания для локаций:

String[] descriptions = {"Пустыня морозов", "Пустыня солнца", "Пустыня пропавших душ",  
 "Пустыня искателей золота", "Зимний лес", "Зеленый лес", "Лес загадок", "Волшебный лес",  
 "Пустыня \"Тысяча ночей\"", "Лес некромантов", "Лес духов", "Лес кошмаров", "Лес криков",  
 "Лес \"Вырвиглаз\"", "Пустыня злорадства", "Пустыня мертвецов", "Пустыня пиратов",  
 "Пустыня разбойников", "Лес сказок", "Лес Бабы-Яги"};

Сам список подготовленных загадок (см. Приложение, class Create, метод createRiddle()), создается в начале игры независимо от действий игрока.

Так же программа загружает и сохраняет данные из/в файл(а) SaveLoad.dat.

Сохранение и чтение сохраненных данных содержится в SaveLoad.class, реализуют эти действия соответствующие методы – serializeLocation(…) и deserializeLocation(…).

Как говорилось ранее, в начале игры считываются сохраненные данные. Если они есть игроку будет предложено либо продолжить игру с этими данными либо начать игру заново. Если сохраненных данных нет, то игроку будет предложено начать новою игру. В случае создания новой игры игроку предлагается ввести имя (обновить). Ограничений в имени нет. Игроку автоматически присваивается здоровье уровня 5.

Далее начинается сама игра. В начале каждого нового круга формируется список из 10 загадок, которые берутся из заранее сформированных 24-х загадок, в методе:

public ArrayList<Location> randomLocation(int limitSize, ArrayList<Location> list) {  
 Log.*writeInto*("Create: " + "randomLocation()");  
 ArrayList<Location> locations = (ArrayList<Location>) list.stream()  
 .collect(Collectors.*collectingAndThen*(Collectors.*toList*(), collected -> {  
 Collections.*shuffle*(collected);  
 return collected.stream();  
 }))  
 .limit(limitSize)  
 .collect(Collectors.*toList*());  
 return locations;  
 }

На выходе мы получаем список, который отправляем на сортировку по уровню сложности в метод:

public ArrayList<Location> sortedLocation(ArrayList<Location> list) {  
 Log.*writeInto*("Create: " + "sortedLocation()");  
 return (ArrayList<Location>)  
 list.stream().sorted(Comparator.*comparing*(Location::getLevel)).collect(Collectors.*toList*());  
 }

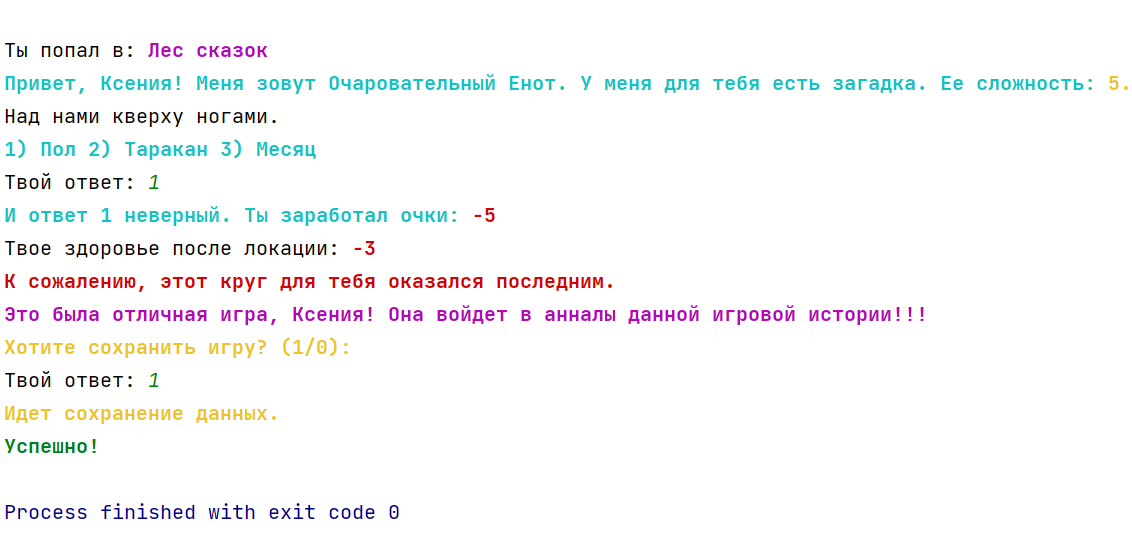
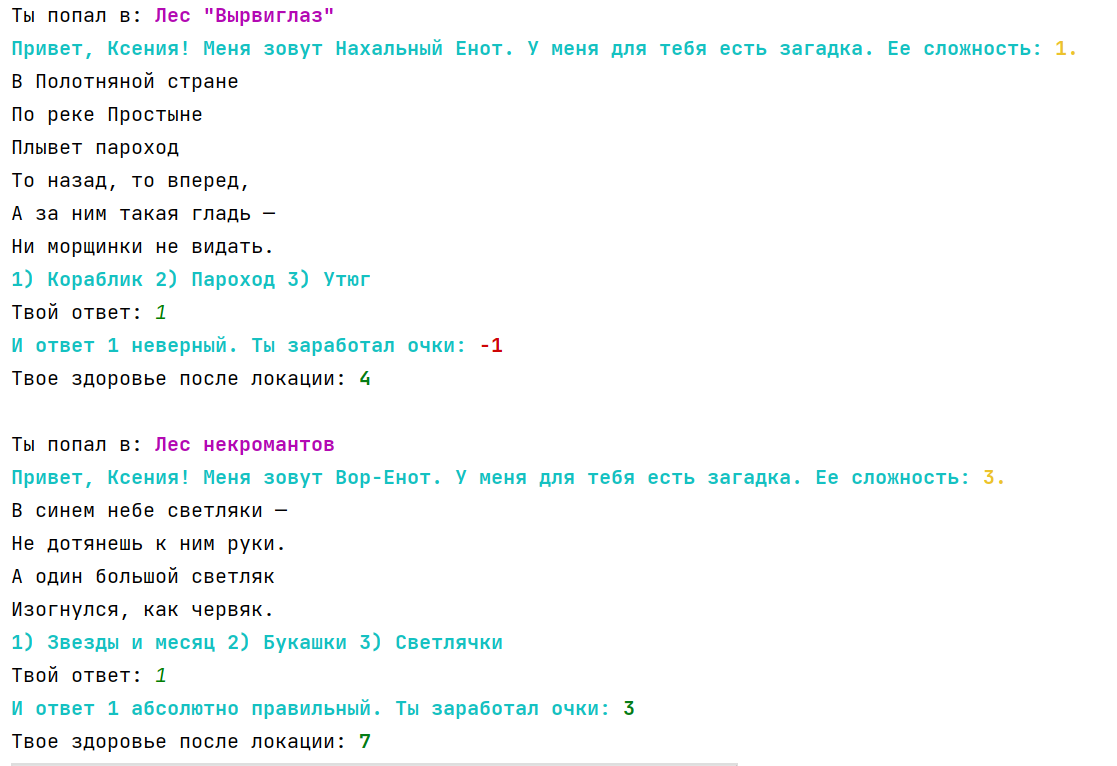
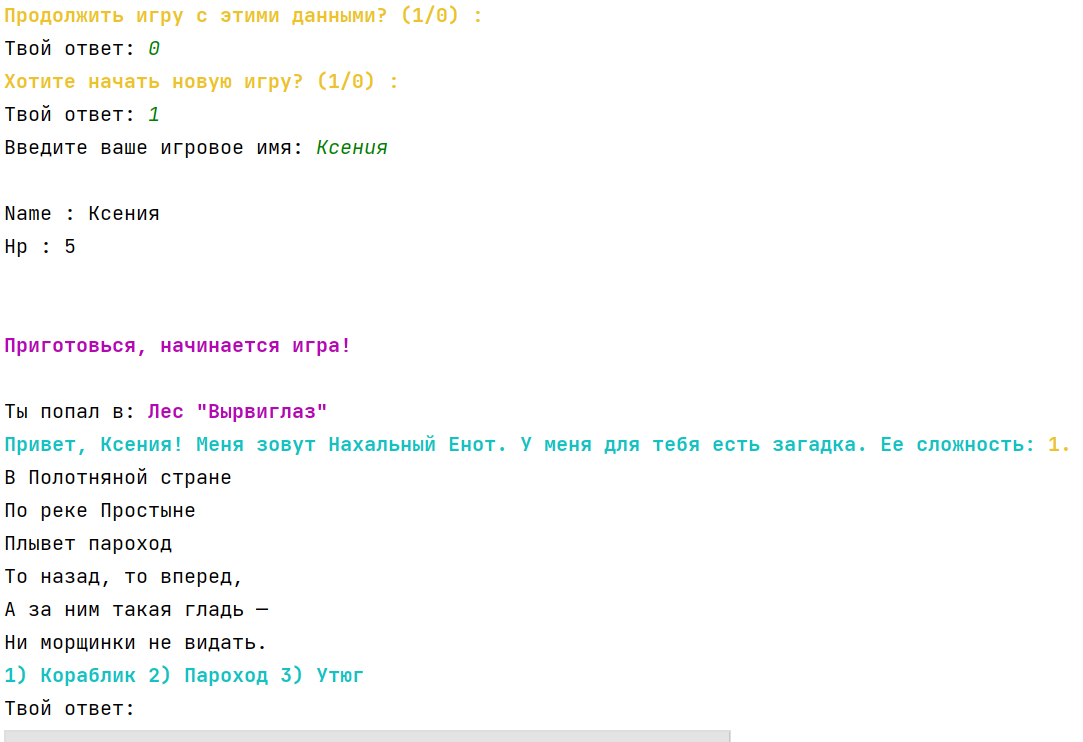
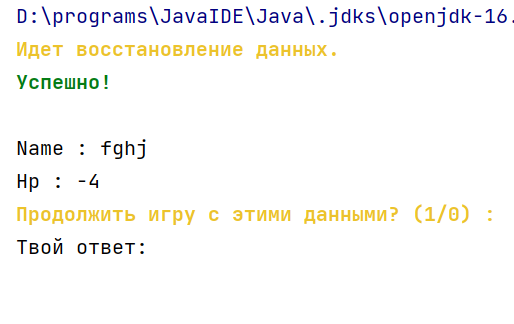
Далее игрок, в порядке очереди, попадает на локацию, где его встречает NPC. Персонаж выдает игроку загадку, если игрок отвечает не верно, то у него отнимается здоровье в размере сложности загадки, если угадывает, то зарабатывает очки здоровья в размере сложности загадки. Если у игрока здоровье становится ровно или ниже нуля, то он покидает игру. Если полностью круг загадок пройден, то игроку предлагается продолжить либо выйти.

При выходе из игры игроку предложат сохранить данные.

Если он отказывается, то при следующем включении программы игроку будет предложено использовать последние сохранные данные.

**Вывод приложения.**

Примерный вывод приложения.



**Приложение.**

**Class GameLoop**

public class GameLoop {  
  
 static public ArrayList<Location> *locations* = new ArrayList<>();  
 static public Player *player* = new Player();  
  
 public static void main(String[] args) {  
 AutoSaveLogWorker autoSaveLogWorker = new AutoSaveLogWorker();  
 Thread thread = new Thread(autoSaveLogWorker);  
 thread.start();  
  
 Create create = new Create();  
 boolean save = *deserializeLocation*();  
 ArrayList<Location> list = create.createLocation();  
 boolean flag = false;  
  
 if(save) {  
 flag = true;  
 View.*printlnMessage*(*player*.toString());  
 flag = *getAnswer*(*BrightYellow\_BOLD* + "Продолжить игру с этими данными? (1/0) :" + *RESET*);  
 }  
 if(!flag) {  
 flag = *getAnswer*(*BrightYellow\_BOLD* + "Хотите начать новую игру? (1/0) :" + *RESET*);  
 if (flag) {  
 *player* = create.createPlayer();  
 *locations* = create.randomLocation(10, list);  
 *locations* = create.sortedLocation(*locations*);  
  
 View.*printlnMessage*(*player*.toString());  
 } else {  
 *endGame*();  
 }  
 }  
  
 flag = *player*.isDead(0);  
 while (flag) {  
 View.*printlnMessage*(*MAGENTA\_BOLD* + "\n\nПриготовься, начинается игра!" + *RESET*);  
 if(!save) {  
 *locations* = create.randomLocation(10, list);  
 *locations* = create.sortedLocation(*locations*);  
 }  
  
 for(int i = 0; i < 10 && flag; i++) {  
 Log.*writeInto*("GameLoop: " + "main(), i = " + i);  
 View.*printlnMessage*("\nТы попал в: " + *MAGENTA\_BOLD* + *locations*.get(i).getDescription() + *RESET*);  
 flag = *locations*.get(i).getNpc().toTalk();  
 View.*printlnMessage*("Твое здоровье после локации: " +  
 (flag ? *GREEN\_BOLD* : *RED\_BOLD*) + *player*.getHp() + *RESET*);  
 }  
  
 if(flag) {  
 View.*printlnMessage*(*GREEN\_BOLD* +  
 "Ты прошел очередной круг загадок и до сих пор жив, мои поздравления!" + *RESET*);  
 flag = *getAnswer*(*BrightYellow\_BOLD* + "Хочешь продолжить ? (1/0) : " + *RESET*);  
 }  
 else {  
 View.*printlnMessage*(*RED\_BOLD* + "К сожалению, этот круг для тебя оказался последним." + *RESET*);  
 }  
 save = false;  
 }  
  
 *endGame*();  
 }  
  
 public static boolean getAnswer(String message) {  
 boolean flag = true;  
 while (true) {  
 View.*printlnMessage*(message);  
 int answer = Input.*inputAnswer*();  
 if (answer == 0) {  
 flag = false;  
 break;  
 } else if (answer == 1) {  
 flag = true;  
 break;  
 } else {  
 View.*printlnMessage*("Не пойму твой ответ, попробуй еще раз.");  
 }  
 }  
 return flag;  
 }  
  
 public static void endGame() {  
 Log.*writeInto*("GameLoop" + ": endGame.");  
 View.*printlnMessage*(*MAGENTA\_BOLD* + "Это была отличная игра, " + *player*.getName() +  
 "! Она войдет в анналы данной игровой истории!!!" + *RESET*);  
 boolean flag = *getAnswer*(*BrightYellow\_BOLD* + "Хотите сохранить игру? (1/0): " + *RESET*);  
 if(flag) *serializeLocation*(*player*, *locations*);  
 System.*exit*(0);  
 }  
  
}

**class Create**

public class Create {  
  
 String[] nameNPC = {"Обычный Лис", "Пушистый Енот", "Красный Лис", "Пустынный Лис",  
 "Лесной Лис", "Загодочный Лис", "Игривый Енот", "Зеленый Лис", "Волшеный Лис", "Редкий Лис",  
 "Разбойник-Енот", "Вор-Енот", "Дружелюбный Енот", "Нахальный Енот", "Очаровательный Енот"};  
  
 String[] descriptions = {"Пустыня морозов", "Пустыня солнца", "Пустыня пропавших душ",  
 "Пустыня искателей золота", "Зимний лес", "Зеленый лес", "Лес загадок", "Волшебный лес",  
 "Пустыня \"Тысяча ночей\"", "Лес некромантов", "Лес духов", "Лес кошмаров", "Лес криков",  
 "Лес \"Вырвиглаз\"", "Пустыня злорадства", "Пустыня мертвецов", "Пустыня пиратов",  
 "Пустыня разбойников", "Лес сказок", "Лес Бабы-Яги"};  
  
 ArrayList<Riddle> riddles = new ArrayList<>();  
  
 public Create() {  
 createRiddle();  
 }  
  
 private void createRiddle() {  
 Log.*writeInto*("Create: " + "createRiddle()");  
 Random rnd = new Random();  
 riddles.add(new Riddle(rnd.nextInt(4 - 1) + 1,  
 "В Полотняной стране\n" +  
 "По реке Простыне\n" +  
 "Плывет пароход\n" +  
 "То назад, то вперед,\n" +  
 "А за ним такая гладь —\n" +  
 "Ни морщинки не видать.",  
 "Утюг",  
 new String[]{"Кораблик", "Пароход"}));  
 riddles.add(new Riddle(rnd.nextInt(4 - 1) + 1,  
 "В брюшке — баня,\n" +  
 "В носу — решето,\n" +  
 "Нос — хоботок,\n" +  
 "На голове — пупок,\n" +  
 "Всего одна рука\n" +  
 "Без пальчиков,\n" +  
 "И та — на спине\n" +  
 "Калачиком.",  
 "Чайник",  
 new String[]{"Самовар", "Кувшин"}));  
 riddles.add(new Riddle(rnd.nextInt(4 - 1) + 1,  
 "Стоит дуб,\n" +  
 "В нем двенадцать гнезд,\n" +  
 "В каждом гнезде\n" +  
 "По четыре яйца,\n" +  
 "В каждом яйце\n" +  
 "По семи цыпленков.",  
 "Год",  
 new String[]{"Птицефабрика", "Дуб"}));  
 riddles.add(new Riddle(rnd.nextInt(4 - 1) + 1,  
 "В синем небе светляки —\n" +  
 "Не дотянешь к ним руки.\n" +  
 "А один большой светляк\n" +  
 "Изогнулся, как червяк.",  
 "Звезды и месяц",  
 new String[]{"Букашки", "Светлячки"}));  
 riddles.add(new Riddle(rnd.nextInt(4 - 1) + 1,  
 "Вдруг из черной темноты\n" +  
 "В небе выросли кусты.\n" +  
 "А на них-то голубые,\n" +  
 "Пунцовые, золотые\n" +  
 "Распускаются цветы\n" +  
 "Небывалой красоты.\n" +  
 "И все улицы под ними\n" +  
 "Тоже стали голубыми,\n" +  
 "Пунцовыми, золотыми,\n" +  
 "Разноцветными.",  
 "Салют",  
 new String[]{"Звезды", "Кусты малины"}));  
 riddles.add(new Riddle(rnd.nextInt(4 - 1) + 1,  
 "Вот иголки и булавки\n" +  
 "Выползают из-под лавки,\n" +  
 "На меня они глядят,\n" +  
 "Молока они хотят.",  
 "Ёж",  
 new String[]{"Иголки для шитья", "Булавки для шитья"}));  
 riddles.add(new Riddle(rnd.nextInt(4 - 1) + 1,  
 "Страну чудес откроем мы\n" +  
 "И встретимся с героями\n" +  
 "В строчках,\n" +  
 "На листочках,\n" +  
 "Где станции на точках.",  
 "Книга",  
 new String[]{"Кино", "Телефонная книга"}));  
 riddles.add(new Riddle(rnd.nextInt(4 - 1) + 1,  
 "Ёжик странный у Егорки\n" +  
 "На окне сидит в ведерке.\n" +  
 "День и ночь он дремлет,\n" +  
 "Спрятав ножки в землю.",  
 "Кактус",  
 new String[]{"Ёж", "Иголки для шитья"}));  
  
  
 riddles.add(new Riddle(rnd.nextInt(7 - 5) + 5,  
 "Стучит,\n" +  
 "Гремит,\n" +  
 "Вертится,\n" +  
 "Ходит весь\n" +  
 "Свой век,\n" +  
 "А не человек.",  
 "Настенные часы",  
 new String[]{"Корманные часы", "Наручные часы"}));  
 riddles.add(new Riddle(rnd.nextInt(7 - 5) + 5,  
 "Кафтан на мне зеленый,\n" +  
 "А сердце — как кумач;\n" +  
 "На вкус, как сахар, сладок,\n" +  
 "На вид — похож на мяч.",  
 "Арбуз",  
 new String[]{"Персик", "Дыня"}));  
 riddles.add(new Riddle(rnd.nextInt(7 - 5) + 5,  
 "Что загадка эта значит?\n" +  
 "Ничего я не пойму:\n" +  
 "По листве зайчонок скачет\n" +  
 "И рассеивает тьму.",  
 "Солнечный свет",  
 new String[]{"Зайчик", "Ветерок"}));  
 riddles.add(new Riddle(rnd.nextInt(7 - 5) + 5,  
 "Наш зверок\n" +  
 "С вершок.\n" +  
 "Носом шмыг, шмыг,\n" +  
 "Хвостиком дрыг, дрыг,\n" +  
 "А дело делает.",  
 "Иголка с ниткой",  
 new String[]{"Ёж", "Щенок"}));  
 riddles.add(new Riddle(rnd.nextInt(7 - 5) + 5,  
 "Загадка у меня к вам есть,\n" +  
 "Загадка непростая:\n" +  
 "Какую букву можно есть,\n" +  
 "Когда она пустая?",  
 "Капуста",  
 new String[]{"Арбуз", "Яблоко"}));  
 riddles.add(new Riddle(rnd.nextInt(7 - 5) + 5,  
 "Над нами кверху ногами.",  
 "Таракан",  
 new String[]{"Месяц", "Пол"}));  
 riddles.add(new Riddle(rnd.nextInt(7 - 5) + 5,  
 "Горит столб, а углей нет.",  
 "Свеча",  
 new String[]{"Загоревшийся дом", "Костер инквизиции"}));  
 riddles.add(new Riddle(rnd.nextInt(7 - 5) + 5,  
 "Под стеклом сижу, в одну сторону гляжу.",  
 "Портрет",  
 new String[]{"Таракан", "Пыль"}));  
  
  
 riddles.add(new Riddle(rnd.nextInt(10 - 8) + 8,  
 "В лесу без огня котел кипит.",  
 "Муравейник",  
 new String[]{"Потухший котел", "Пожар"}));  
 riddles.add(new Riddle(rnd.nextInt(10 - 8) + 8,  
 "Дом открыт\n" +  
 "со всех сторон.\n" +  
 "В доме —\n" +  
 "Тысячи колонн.\n" +  
 "Над колоннами —\n" +  
 "Шатры.\n" +  
 "Под колоннами —\n" +  
 "Ковры.\n" +  
 "Там живут —\n" +  
 "И в коврах,\n" +  
 "И в колоннах,\n" +  
 "И в шатрах.",  
 "Лес",  
 new String[]{"Цирк", "Многоэтажка"}));  
 riddles.add(new Riddle(rnd.nextInt(10 - 8) + 8,  
 "Нет ушей, а слышит. Нет рук, а пишет.",  
 "Магнитофон",  
 new String[]{"Стукач", "Инвалид"}));  
 riddles.add(new Riddle(rnd.nextInt(10 - 8) + 8,  
 "Бычок рогат, в руках зажат. Еду хватает, а сам голодает.",  
 "Ухват",  
 new String[]{"Рука", "Бычок"}));  
 riddles.add(new Riddle(rnd.nextInt(10 - 8) + 8,  
 "Что проходит большое пространство, не двигаясь с места?",  
 "Дорога",  
 new String[]{"Время", "Телефонный разговор"}));  
 riddles.add(new Riddle(rnd.nextInt(10 - 8) + 8,  
 "Чем больше отдаю, тем больше вырастаю. Величину свою отдачей измеряю.",  
 "Яма",  
 new String[]{"Карьерный рост", "Личностный рост"}));  
 riddles.add(new Riddle(rnd.nextInt(10 - 8) + 8,  
 "Сила не сила, а грязь убило.",  
 "Мыло",  
 new String[]{"Спирт", "Дезинфектор"}));  
 riddles.add(new Riddle(rnd.nextInt(10 - 8) + 8,  
 "Сделана из тряпок, а ниток не видно.",  
 "Бумага",  
 new String[]{"Кофта", "Салфетка"}));  
 }  
  
 public ArrayList<Location> createLocation() {  
 Log.*writeInto*("Create: " + "createLocation()");  
 Random rnd = new Random();  
 ArrayList<Location> locations = new ArrayList<>();  
 for(int i = 0; i < riddles.size(); i++) {  
 String des = descriptions[rnd.nextInt(descriptions.length)];  
 NPC npc = createNPC(i);  
 int level = npc.getRiddle().getComplexity();  
 locations.add(new Location(npc, level, des));  
 }  
 return locations;  
 }  
  
 public Player createPlayer() {  
 String name = Input.*inputName*();  
 return new Player(name, 5);  
 }  
  
 private NPC createNPC(int i) {  
 Random rnd = new Random();  
 String name = nameNPC[rnd.nextInt(nameNPC.length)];  
 Riddle riddle = riddles.get(i);  
 return new NPC(name, riddle);  
 }  
  
 public ArrayList<Location> randomLocation(int limitSize, ArrayList<Location> list) {  
 Log.*writeInto*("Create: " + "randomLocation()");  
 ArrayList<Location> locations = (ArrayList<Location>) list.stream()  
 .collect(Collectors.*collectingAndThen*(Collectors.*toList*(), collected -> {  
 Collections.*shuffle*(collected);  
 return collected.stream();  
 }))  
 .limit(limitSize)  
 .collect(Collectors.*toList*());  
 return locations;  
 }  
  
 public ArrayList<Location> sortedLocation(ArrayList<Location> list) {  
 Log.*writeInto*("Create: " + "sortedLocation()");  
 return (ArrayList<Location>)  
 list.stream().sorted(Comparator.*comparing*(Location::getLevel)).collect(Collectors.*toList*());  
 }  
}

**class Input**

public class Input {  
  
 public static String inputName() {  
 View.*printMessage*("Введите ваше игровое имя: ");  
 Scanner input = new Scanner(System.*in*);  
 String name = input.nextLine();  
 return name;  
 }  
  
 public static int inputAnswer() {  
 while (true) {  
 View.*printMessage*("Твой ответ: ");  
 try {  
 Scanner input = new Scanner(System.*in*);  
 int number = input.nextInt();  
 return number;  
 } catch (InputMismatchException e) {  
 View.*printlnMessage*(View.*BrightYellow\_BOLD* + "Введено не число, попробуйте заново." + View.*RESET*);  
 }  
 }  
 }  
}

**class View**

public class View {  
 public static final String *RESET* = "\033[0m";  
 public static final String *GREEN\_BOLD* = "\033[1;32m";  
 public static final String *RED\_BOLD* = "\033[1;31m";  
 public static final String *MAGENTA\_BOLD* = "\033[1;35m";  
 public static final String *BrightCyan\_BOLD* = "\033[1;96m";  
 public static final String *BrightYellow\_BOLD* = "\033[1;93m";  
  
  
 public static void printlnMessage(String message) {  
 System.*out*.println(message);  
 }  
  
 public static void printMessage(String message) {  
 System.*out*.print(message);  
 }  
}

**class Location**

public class Location implements Serializable {  
  
 private NPC npc;  
 private int level;  
 private String description;  
  
 public Location(NPC npc, int level, String description) {  
 this.npc = npc;  
 this.level = level;  
 this.description = description;  
 }  
  
 public Location() {  
 this.npc = new NPC();  
 this.level = 0;  
 this.description = "";  
 }  
  
 public NPC getNpc() {  
 return npc;  
 }  
  
 public void setNpc(NPC npc) {  
 this.npc = npc;  
 }  
  
 public int getLevel() {  
 return level;  
 }  
  
 public void setLevel(int level) {  
 this.level = level;  
 }  
  
 public String getDescription() {  
 return description;  
 }  
  
 public void setDescription(String description) {  
 this.description = description;  
 }  
  
 @Override  
 public String toString() {  
 return new StringBuffer(*MAGENTA\_BOLD* + "\nDescription : " + *RESET*).append(this.description)  
 .append(*MAGENTA\_BOLD* + "\nLevel : " + *RESET*).append(this.level)  
 .append(*MAGENTA\_BOLD* + "\nNPC : " + *RESET*).append(this.npc).toString();  
 }  
}

**class Player**

public class Player implements Serializable {  
  
 private String name;  
 private int hp;  
  
 public Player(String name, int hp) {  
 this.name = name;  
 this.hp = hp;  
 }  
  
 public Player() {  
 this.name = "Name";  
 this.hp = 0;  
 }  
  
 public String getName() {  
 return name;  
 }  
  
 public void setName(String name) {  
 this.name = name;  
 }  
  
 public int getHp() {  
 return hp;  
 }  
  
 public void setHp(int hp) {  
 this.hp = hp;  
 }  
  
 public boolean isDead(int result) {  
 hp += result;  
 return hp > 0;  
 }  
  
 @Override  
 public String toString() {  
 return new StringBuffer("\nName : ").append(this.name)  
 .append("\nHp : ").append(this.hp).toString();  
 }  
}

**class NPC**

public class NPC implements Serializable {  
  
 private String name;  
 private Riddle riddle;  
  
 public NPC(String name, Riddle riddle) {  
 this.name = name;  
 this.riddle = riddle;  
 }  
  
 public NPC() {  
 this.name = "Name";  
 }  
  
 public String getName() {  
 return name;  
 }  
  
 public void setName(String name) {  
 this.name = name;  
 }  
  
 public Riddle getRiddle() {  
 return riddle;  
 }  
  
 public void setRiddle(Riddle riddle) {  
 this.riddle = riddle;  
 }  
  
 public boolean toTalk() {  
 Log.*writeInto*("NPC: " + "toTalk()");  
 View.*printlnMessage*(*BrightCyan\_BOLD* + "Привет, " + GameLoop.*player*.getName() + "!" +  
 " Меня зовут " + getName() + ". У меня для тебя есть загадка. Ее сложность: " + *RESET* +  
 *BrightYellow\_BOLD* + riddle.getComplexity() + "." + *RESET*);  
 View.*printlnMessage*(riddle.getRiddle());  
 View.*printlnMessage*(*BrightCyan\_BOLD* + riddle.getAllAnswers() + *RESET*);  
  
 int answer = Input.*inputAnswer*();  
  
 if(answer == riddle.getTrueNumber()) {  
 View.*printlnMessage*(*BrightCyan\_BOLD* + "И ответ " + answer + " абсолютно правильный." +  
 " Ты заработал очки: " + *RESET* + *GREEN\_BOLD* + riddle.getComplexity() + "" + *RESET*);  
 return GameLoop.*player*.isDead(riddle.getComplexity());  
 }  
 else {  
 View.*printlnMessage*(*BrightCyan\_BOLD* + "И ответ " + answer + " неверный." +  
 " Ты заработал очки: " + *RESET* + *RED\_BOLD* + -riddle.getComplexity() + "" + *RESET*);  
 return GameLoop.*player*.isDead(-riddle.getComplexity());  
 }  
 }  
  
  
 @Override  
 public String toString() {  
 return new StringBuffer("\nName : ").append(this.name)  
 .append(*MAGENTA\_BOLD* + "\nRiddle : " + *RESET*).append(this.riddle).toString();  
 }  
}

**class Riddle**

public class Riddle implements Serializable {  
  
 private String riddle;  
 private String trueAnswer;  
 private String[] falseAnswers;  
 private int complexity;  
 private int trueNumber;  
  
 public Riddle(int complexity, String riddle, String trueAnswer, String[] falseAnswers) {  
 this.complexity = complexity;  
 this.riddle = riddle;  
 this.trueAnswer = trueAnswer;  
 this.falseAnswers = falseAnswers;  
 }  
  
 public int getTrueNumber() {  
 return trueNumber;  
 }  
  
 public int getComplexity() {  
 return complexity;  
 }  
  
 public void setComplexity(int complexity) {  
 this.complexity = complexity;  
 }  
  
 public String getRiddle() {  
 return riddle;  
 }  
  
 public void setRiddle(String riddle) {  
 this.riddle = riddle;  
 }  
  
 public String getTrueAnswer() {  
 return trueAnswer;  
 }  
  
 public void setTrueAnswer(String trueAnswer) {  
 this.trueAnswer = trueAnswer;  
 }  
  
 public String[] getFalseAnswers() {  
 return falseAnswers;  
 }  
  
 public void setFalseAnswers(String[] falseAnswers) {  
 this.falseAnswers = falseAnswers;  
 }  
  
 public String getAllAnswers() {  
 Log.*writeInto*("Riddle: " + "getAllAnswers()");  
 List<String> words = new ArrayList<>();  
 words.add(trueAnswer);  
 words.addAll(Arrays.*stream*(falseAnswers).toList());  
 Collections.*shuffle*(words);  
 for(int i = 0; i < 3; ++i) {  
 if(words.get(i).equals(trueAnswer))  
 trueNumber = i + 1;  
 }  
 String s = "1) " + words.get(0) + " 2) " + words.get(1) + " 3) " + words.get(2);  
 return s;  
 }  
  
 @Override  
 public String toString() {  
 return new StringBuffer("\nRiddle : ").append(this.riddle)  
 .append("\nComplexity : ").append(this.complexity)  
 .append("\nTrue Answer : ").append(this.trueAnswer)  
 .append("\nFalse Answers : ").append(this.falseAnswers[0]).append(", ")  
 .append(this.falseAnswers[1]).toString();  
 }  
}

**class SaveLoad**

public class SaveLoad {  
  
 private static final String *fileName* = "SaveLoad.dat";  
  
 public static void serializeLocation(Player player, ArrayList<Location> locations) {  
 Log.*writeInto*("SaveLoad: " + "serializeLocation()");  
 View.*printlnMessage*(*BrightYellow\_BOLD* + "Идет сохранение данных." + *RESET*);  
 try (ObjectOutputStream oos =  
 new ObjectOutputStream(new FileOutputStream(*fileName*))) {  
  
 oos.writeObject(player);  
 oos.writeObject(locations);  
 View.*printlnMessage*(*GREEN\_BOLD* + "Успешно!" + *RESET*);  
  
 } catch (Exception ex) {  
 Log.*writeInto*("SaveLoad: " + "serializeLocation(). ERROR" + ex.getMessage());  
 View.*printlnMessage*(*RED\_BOLD* + ex.getMessage() + *RESET*);  
 }  
 }  
  
 public static boolean deserializeLocation() {  
 Log.*writeInto*("SaveLoad: " + "deserializeLocation()");  
 View.*printlnMessage*(*BrightYellow\_BOLD* + "Идет восстановление данных." + *RESET*);  
 try(ObjectInputStream ois = new ObjectInputStream(new FileInputStream(*fileName*)))  
 {  
 Player player = (Player) ois.readObject();  
 ArrayList<Location> locations=((ArrayList<Location>)ois.readObject());  
 View.*printlnMessage*(*GREEN\_BOLD* + "Успешно!" + *RESET*);  
  
 GameLoop.*player* = player;  
 GameLoop.*locations* = locations;  
 }  
 catch (FileNotFoundException noFile) {  
 Log.*writeInto*("SaveLoad: " + "deserializeLocation(). File not find - " + *fileName*);  
 View.*printlnMessage*(*BrightYellow\_BOLD* + "Не обнаружено сохраненных данных." + *RESET*);  
 return false;  
 }  
 catch(Exception ex){  
 Log.*writeInto*("SaveLoad: " + "serializeLocation(). ERROR" + ex.getMessage());  
 View.*printlnMessage*(*RED\_BOLD* + ex.getMessage() + *RESET*);  
 return false;  
 }  
  
 return true;  
 }  
}

**class AutoSaveLogWorker**

public class AutoSaveLogWorker implements Runnable {  
 @Override  
 public void run() {  
 while (!Thread.*currentThread*().isInterrupted()) {  
 try (FileWriter logWriter = new FileWriter("Log.txt");  
 BufferedWriter bufferedWriter = new BufferedWriter(logWriter)  
 ) {  
 ArrayList<String> logBuffer = Log.*getLogHistory*();  
 for (int i = 0; i < logBuffer.size(); ++i) {  
 String s = logBuffer.get(i);  
 s += "\n";  
 bufferedWriter.write(s);  
 }  
 TimeUnit.*SECONDS*.sleep(5);  
 } catch (IOException | InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
 }  
}

**class Log**

public class Log {  
  
 private static ArrayList<String> *logHistory* = new ArrayList<>();  
  
 public static void writeInto(String message){  
 *logHistory*.add(message);  
 }  
  
 public static ArrayList<String> getLogHistory(){  
 return *logHistory*;  
 }  
  
}