proga

```
# Deepseek
// Новый метод для поедания случайной еды
   public void eatRandomFood() {
       Food randomFood = EventMaker.getFood();
       System.out.println(name + " ест " + randomFood.toString());

       double benefit = randomFood.getBenefit();
       this.HP += benefit;

       if (benefit > 0) {
           System.out.println(name + " чувствует себя лучше! Здоровье +" +
benefit);
       } else if (benefit < 0) {
           System.out.println(name + " чувствует себя хуже! Здоровье " + benefit);
       } else {
           System.out.println(name + " не чувствует изменений");
       }

       // Проверка на смерть от отравления
       if (this.HP <= 0) {
           this.die();
       } else if (this.HP > MAX_HP) {
           this.HP = MAX_HP;
       }
   }

   // Дополнительный метод для поедания конкретной еды
   public void eat(Food food) {
       System.out.println(name + " ест " + food.toString());

       double benefit = food.getBenefit();
       this.HP += benefit;

       if (benefit > 0) {
           System.out.println(name + " чувствует себя лучше! Здоровье +" +
benefit);
       } else if (benefit < 0) {
           System.out.println(name + " чувствует себя хуже! Здоровье " + benefit);
       } else {
           System.out.println(name + " не чувствует изменений");
       }
```

```
        // Проверка на смерть от отравления
        if (this.HP <= 0) {
            this.die();
        } else if (this.HP > MAX_HP) {
            this.HP = MAX_HP;
        }
    }
```

```
# Qwen
 // Вспомогательный метод для вызова функционала поедания еды

    public void eatRandomFood() {

        new FoodEater().eat();

    }



    // Внутренний класс, реализующий логику поедания случайной еды

    private class FoodEater {

        public void eat() {

            // Проверка: если человек мёртв, не может есть

            if (deathTime != null) {

                System.out.println("Human " + name + " мертв и не может есть.");

                return;

            }



            // Получаем случайную еду

            Food food = EventMaker.getFood();

            double benefit = food.getBenefit();

            // Выводим информацию о действии
```

```
            System.out.println(name + " съел(а) " + food + " и получил(а) " +
benefit + " HP.");

            // Применяем эффект еды к здоровью

            HP += benefit;

            // Проверка на превышение максимума HP

            if (HP > MAX_HP) {

                HP = MAX_HP;

            }

            // Проверка на смерть от отрицательного эффекта

            if (HP <= 0) {

                die();

            }

        }

    }
```

```
# мой код
public void eatRandomFood() {
    if (HP == 0) {
        System.out.printf("Human %s не может есть, так как он умир(((%n", name);
        return ;
    }
    Food food = EventMaker.getFood();
    double benefit = food.getBenefit();
    if (benefit > 0) {
        System.out.printf("Human %s съел %s и восполнил %.2f здоровья%n", name,
food, benefit);
        HP = HP + benefit;
    } else {
        System.out.printf("Human %s съел %s и потерял %.2f здоровья%n", name, food,
-benefit);
        if (getLuck() > 57 && HP + benefit <= 0) {
            System.out.println("у Human" + name + " срабатывает удача, и он не
```

```
умирает!!! Его HP равно 0.01");
            HP = 0.01;
        } else {
            die();
        }
    }
}
```

Анализ

	DeepSeek	Qwen	Я
Проверка мертв ли персонаж	нет	есть	есть
HP > MAX_HP	ограничивает	ограничивает	нет ограничений
Вывод сообщений	подробно	кратко	самый богатый
Повторяемость кода	есть	нет	нет
Оригинальные механики	нет	нет	да – luck

Вывод

После сравнения трёх вариантов я вижу картину так.

Мой метод получился самым интересным в игровом плане: я добавил механику удачи, эффект неожиданности, и сама логика ощущается «живой». Но при этом я допустил логическую ошибку — смерть вызывается в ситуации, когда HP может оставаться положительным. Ещё я не обрезаю здоровье по MAX_HP, что может привести к несогласованному состоянию.

Вариант DeepSeek выглядит самым безопасным и технически корректным. Он проверяет переполнение HP, корректно обрабатывает смерть, но в нём много повторяющегося кода, что снижает поддерживаемость.

Решение Qwen оказалось самым странным: внутренняя реализация через отдельный класс выглядит усложнением ради усложнения и не даёт ощутимых преимуществ.