**Задача 10**

Цели задания

* Создать собственные обобщённые классы.
* Закрепить на практике работу с data-классами, sealed-классами, enum-классами, расширениями.
* Применить принципы ООП и взаимодействия между различными объектами для решения практической задачи.

Что нужно сделать

Необходимо разработать консольную программу, которая имитирует сражение между двумя командами, отслеживает ход сражения и собирает информацию о нём.

Команда состоит из воинов, каждый из которых имеет основные характеристики и оружие в зависимости от того, к какому классу воинов он принадлежит. Например: солдат или капитан.

Сражение происходит пошагово: каждый воин из команды атакует противника из другой команды, а также может быть атакован на следующий ход противника.

Дополнительные правила сражения и критерии победы одной из команд сформулируйте и реализуйте самостоятельно. Программа должна собирать необходимую информацию и отображать в консоли текущий ход сражения.

В этом домашнем задании вам предстоит собрать воедино полученные ранее знания из всех предыдущих модулей и превратить их в практические умения. Поскольку работа генерализует и закрепляет материал нескольких модулей, её выполнение может занять больше времени, чем обычно. При этом у вас получится сложный, целостный продукт, максимально приближенный к реальной рабочей задаче. В дальнейшем эту программу можно расширять за счёт нового функционала, а также отрабатывать на ней будущие темы.

Часть 1

Для выполнения задания необходимо разработать дополнительный Generic-класс и использовать его в основной программе.

Одной из стандартных коллекций в программировании является Stack. Эта коллекция характеризуется тем, что хранит элементы в упорядоченном виде по принципу Last in first out — последний вошёл, первый вышел. То есть первый добавленный в стек элемент извлекается из него в последнюю очередь.

**Замечание:** в стандартной библиотеке классов Kotlin данная коллекция не представлена. Зато она реализована в Java, а поскольку мы рассматриваем Kotlin JVM, то к этой коллекции есть доступ.

1. Создайте Generic-класс Stack<T>, который будет хранить элементы согласно правилам коллекции Stack.
2. Создайте в этом классе функции:
   1. push(item: T) — добавляет элемент в стек.
   2. pop(): T? — извлекает элемент из стека. Если стек пустой — возвращает null.
   3. isEmpty(): Boolean — проверяет стек на наличие элементов.
3. Проверьте, что класс работает корректно в соответствии с правилами работы стека.

Часть 2

Разработайте программу:

* Пользователь вводит количество воинов в команде.
* Начинается сражение.
* Сражение происходит автоматически и пошагово до победы одной из команд или ничьей.
* Каждый ход собирается и выводится текущая информация о состоянии сражения.
* По окончании сражения выводится команда-победитель либо ничья.

Для этого вам потребуются следующие компоненты:

    1. Функция-расширение для типа Int. Она должна вычислять, реализовался ли шанс, и возвращать Boolean. Используйте эту функцию для проверок, связанных с вероятностью. Функцию необходимо реализовать с помощью вызова Random.nextInt().

    2. Перечисление (enum) Ammo для видов патронов.

Свойства:

* урон,
* шанс критического урона,
* коэффициент критического урона.

Методы:

* получение текущего урона (рассчитывает текущий урон в соответствии с шансом критического урона и коэффициентом критического урона).

**Критический урон** — существенно больший урон, который наносится с определённой вероятностью. Рассчитывается по формуле **Критический урон = коэффициент \* урон**.

Каждый патрон имеет возможность нанести критический урон. Этот шанс представлен свойством **«Шанс критического урона»**. Например, шанс критического урона 25% означает, что с вероятностью 25% патрон нанесёт урон, равный**урон \* коэффициент**.

В перечислении должно быть минимум три вида патронов с разными параметрами. Чтобы проверить, является ли урон критическим, используйте функцию-расширение из пункта 1.

    3. Sealed-класс для вида стрельбы FireType.

Создайте наследников:

* одиночный выстрел (singleton object),
* стрельба очередями (data class, в конструкторе принимает параметр «размер очереди»).

Создайте дополнительные типы стрельбы при необходимости.

    4. Класс оружия AbstractWeapon.

Свойства:

* Максимальное количество патронов в магазине.
* Вид стрельбы (FireType).
* Магазин патронов. Используйте класс Stack из первой части домашнего задания. При создании оружия магазин пуст.
* Факт наличия патронов в магазине.

Методы:

* создание патрона необходимого типа;
* перезарядка — создаётся новый магазин и заполняется патронами с помощью функции создания патрона;
* получение патронов для выстрела — из магазина получаются патроны в соответствии с типом стрельбы для выстрела(-ов) (количество патронов в магазине должно уменьшаться).

    5. Синглтон-объект Weapons, который создаёт разные виды оружия. Объект Weapons должен создавать анонимные объекты AbstractWeapon. Создайте минимум четыре метода для разных видов оружия.

Пример работы: val pistol: AbstractWeapon = Weapons.createPistol().

    6. Интерфейс Warrior.

Свойства:

* isKilled,
* шанс избежать попадания.

Методы:

* атаковать — на вход принимает противника (Warrior),
* получить урон — на вход принимает количество урона.

    7. Абстрактный класс AbstractWarrior, который реализует интерфейс Warrior.

Свойства:

* максимальный уровень здоровья,
* уклонение — шанс избежать попадания,
* точность — вероятность попадания,
* оружие,
* текущий уровень здоровья.

Методы:

* Атаковать (метод интерфейса) — если патронов в оружии нет, перезарядить оружие и пропустить ход. В противном случае получить патроны для выстрела. Для каждого из них проверить, соответствует ли патрон точности война и уклонению противника. Для патронов, которые попадают в противника, рассчитать суммарный урон. Нанести урон врагу.
* Понести урон (метод интерфейса) — отнять от текущего уровня здоровья урон.

    8. Классы для воинов, наследуемых от AbstractWarrior.

Сделайте несколько классов воинов. Разделите их, например, по старшинству. У более старшего воина должен быть больший уровень здоровья, большая точность, лучше оружие. *Примеры классов: General, Captain, Soldier.*

    9. Класс команды Team, которая состоит из воинов.

В команду воины набираются по вероятности: чем выше вероятность, тем ниже ранг. *Пример: с вероятностью 10% создаётся генерал, иначе с вероятностью 40% создаётся капитан, иначе создаётся солдат.*

    10. Sealed-класс BattleState, который описывает текущее состояние сражения.

Существует четыре состояния:

* Прогресс — содержит в себе необходимую информацию для вывода в консоль. Например: суммарное здоровье команды или количество оставшихся воинов. Необходимую информацию определите самостоятельно.
* Победа первой команды.
* Победа второй команды.
* Ничья.

    11. Класс битвы Battle. Этот класс отвечает за логику сражения.

Свойства:

* команда 1,
* команда 2,
* битва завершена.

Методы:

* Получить состояние сражения.

        Определите критерии победы или поражения. Например:

        в каждой команде проверяется, есть ли живые воины. Если у двух команд есть живые воины — возвращается состояние прогресса. Иначе — победа одной из команд.

* Совершить итерацию битвы.

        Определите правила проведения итерации сражения. Например:

        воины двух команд перемешиваются. Воины из каждой команды наносят друг другу урон по очереди, если они живы.

Советы и рекомендации

* Разбивайте код на более мелкие функции, если этого требует ваша программа, даже если это не указано в задании.

        По желанию вы можете добавить в программу дополнительный функционал. В этом случае обязательно сообщите об этом проверяющему преподавателю, чтобы он смог это проверить.

* Чтобы избежать ненужных исправлений на начальных пунктах, настоятельно рекомендуем выполнять домашнее задание поэтапно, консультируясь с преподавателем.
* Вы можете отправлять на проверку частично выполненное задание. Например:
  + Часть 1.
  + Часть 2 пункты 1–5.
  + Часть 2 пункты 6–8.
  + Часть 2 пункты 9–11.
* Если написанная программа по какой-то причине не работает и вы не можете разобраться, в чём дело, — воспользуйтесь [отладчиком](https://medium.com/nuances-of-programming/%D0%B2%D1%81%D1%91-%D1%87%D1%82%D0%BE-%D0%B2%D1%8B-%D1%85%D0%BE%D1%82%D0%B5%D0%BB%D0%B8-%D0%B7%D0%BD%D0%B0%D1%82%D1%8C-%D0%BE%D0%B1-%D0%BE%D1%82%D0%BB%D0%B0%D0%B4%D0%BA%D0%B5-%D0%B2-intellij-idea-f426dcc7cd1b). С помощью точек останова и просмотра информации о состоянии объектов вы сможете выявить ошибку в коде.

Что оценивается

* Программа запускается, работает корректно и выводит всю необходимую информацию.
* Выполнены все пункты задания.
* Соблюдена инкапсуляция. Доступны только публичные члены классов.
* Классы должны быть открытыми для наследования только в случае необходимости.
* Методы должны быть открытыми для переопределения только в случае необходимости.
* Функции соответствуют принципам чёрного ящика, выполняют атомарные операции и независимы друг от друга.
* Функции возвращают неизменяемые коллекции.
* Названия компонентов программы отражают их назначение.
* Переменные, которые не изменяются в программе, должны быть объявлены неизменяемыми.
* Различные выводы в консоль начинаются с новой строки.

*Плохо:* «Вывод информации 1. Вывод информации 2».

*Хорошо:*

«Вывод информации 1.

Вывод информации 2».