ТОМОВ ЩЕ НИ ЕБЕ

**1. Stack (Стек):**

* **Структура и организация**: Stack е организиран като LIFO (Last In, First Out) структура, където последно добавената стойност е първата, която се извлича. Той се използва за управление на методи и локални променливи.
* **Съдържание**: В Stack се съхраняват примитивни типове данни (като int, float, boolean и др.) и референции (адреси) към обекти в паметта.

**2. Heap (Хийп):**

* **Структура и организация**: Heap е неструктурирана област от паметта, където обектите и динамично разпределените променливи се съхраняват. Обектите тук могат да съществуват, докато не бъдат изрично изтрити (или събирани чрез garbage collector).
* **Съдържание**: В Heap се съхраняват обекти и данни от референтни типове (като масиви, класове и т.н.). Референциите към тези обекти се намират в Stack.

3. LIFO (**Last In, First Out**) е принцип за организация на данните, при който последният елемент, добавен в структурата от данни, е първият, който ще бъде изваден. Представете си LIFO като купчина книги: ако сложите една книга върху други, за да вземете книга от купчината, ще вземете най-горната (последно поставената).

Този принцип се използва често в структура от данни, наречена **стек (stack)**.

**Как работи LIFO?**

1. **Добавяне на елемент (Push)**: Когато добавяме нов елемент в LIFO структурата, той се поставя най-отгоре.
2. **Премахване на елемент (Pop)**: Когато премахнем елемент, се изважда този, който е най-отгоре (т.е. последният добавен).

Картина, която съдържа текст, екранна снимка, Шрифт, номер

Описанието е генерирано автоматично

**1. Примитивни типове данни**

Примитивните типове са базовите, прости типове данни, които не могат да бъдат разглобявани на по-малки компоненти. Те директно съдържат стойността на данните и се съхраняват в стека (stack). Примитивните типове са независими и не съдържат други обекти или стойности. Примери за такива типове са:

* **Цели числа (int)**: Например, 1, 42, -10
* **Дробни числа (float, double)**: Например, 3.14, 2.71
* **Булеви стойности (boolean)**: true или false
* **Знаци (char)**: Единични символи, например 'a', 'Z'

**2. Референтни типове данни**

Референтните типове представляват по-сложни структури от данни и се съхраняват в паметта по различен начин. Те не съдържат директно стойността на данните, а референция (указател) към мястото в паметта, където са съхранени самите данни. Референтните типове обикновено се съхраняват в хипа (heap). Примери за такива типове са:

* **Обекти (Object)**: Класове, дефинирани от потребителя
* **Масиви (Array)**: Колекции от стойности, например int[]
* **Стрингове (String)**: Въпреки че може да се мисли за String като примитивен тип, в много езици (като Java), това е референтен тип.
* **Колекции (List, Map, Set и др.)**

**Основни разлики:**

* **Съхранение**: Примитивните типове съхраняват стойностите директно в стека, докато референтните типове съхраняват само референция в стека, а стойностите в хипа.
* **Копиране**: Примитивните типове се копират по стойност, докато референтните типове се копират по референция (адрес).
* **Размер**: Примитивните типове имат фиксиран размер, докато референтните типове могат да заемат променлив обем от памет.
* **Изпълнение**: Операциите с примитивни типове са по-бързи и използват по-малко памет в сравнение с референтните типове.

Картина, която съдържа текст, екранна снимка, номер, Шрифт

Описанието е генерирано автоматично

**1. Съхранение на данни:**

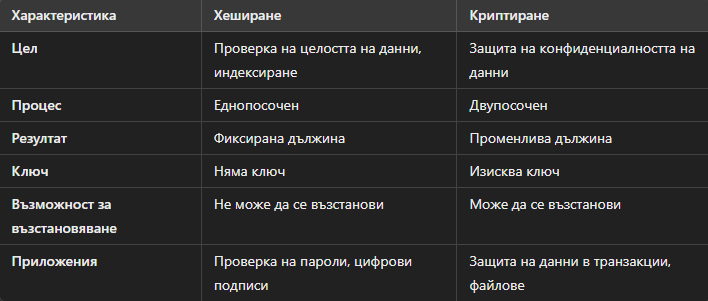
* **ArrayList**:
  + Използва динамичен масив за съхранение на елементите.
  + Елементите са последователно съхранени в паметта, което позволява бърз достъп по индекс (случайно достъпване).
* **LinkedList**:
  + Използва двусвързан списък. Всеки елемент (възел) съдържа данни и указател към следващия и предишния елемент.
  + За достъп до елемент е необходимо да се обходи списъкът, което отнема повече време за операции като достъп по индекс.

**2. Операции за достъп и модификация:**

* **ArrayList**:
  + **Бърз достъп по индекс**: O(1) за четене и промяна на елемент по индекс, тъй като масивът е последователно съхранен.
  + **Бавно добавяне/изтриване**: O(n) при добавяне или премахване на елементи в началото или средата на списъка, защото е необходимо преместване на останалите елементи.
* **LinkedList**:
  + **Бавен достъп по индекс**: O(n) за достъп до елемент, тъй като трябва да се обходи списъкът последователно от началото или края.
  + **Бързо добавяне/изтриване**: O(1) за добавяне и премахване на елементи в началото или края на списъка, тъй като е нужно само да се променят връзките между възлите.

**3. Памет:**

* **ArrayList**:
  + Заема по-малко памет за данните, тъй като съдържа само елементите.
  + При разширяване на капацитета, масивът се копира в нов масив с по-голям размер, което може да отнеме време.
* **LinkedList**:
  + Използва повече памет, защото всеки възел съхранява не само данни, но и връзки към следващия и предишния елемент.
  + Няма нужда от преоразмеряване, защото свързан



**1. Платформена независимост**

Java е проектирана да бъде независима от платформата, което означава, че написаната програма може да работи на всяка система, която има инсталирана Java виртуална машина (JVM). Тази независимост е постигната чрез концепцията за „пишеш веднъж, изпълняваш навсякъде“ (WORA).

**2. Java Virtual Machine (JVM)**

JVM е основният компонент, който изпълнява Java програми. Тя компилира байт-кода на Java в машинен код, който е специфичен за платформата. Основните стъпки в работата на Java включват:

* **Компилиране на Java код**: Когато напишете Java код (файлове с разширение .java), той се компилира с помощта на Java компилатор (javac) в байт-код (файлове с разширение .class).
* **Изпълнение на байт-кода**: JVM интерпретира байт-кода и го изпълнява. Това позволява на Java програмите да работят на всяка платформа, която има инсталирана JVM.

**Примерен работен процес на Java**

1. **Написване на код**: Разработчикът пише Java код в текстов редактор или IDE (например IntelliJ IDEA или Eclipse).
2. **Компилиране**: Кодът се компилира в байт-код с помощта на компилатора (javac).
3. **Изпълнение**: Байт-кодът се изпълнява от JVM, която преобразува байт-кода в машинен код, подходящ за конкретната платформа.