# Структури данни

SAP GEEKYCAMP 4.0

Божин Кацарски
6.09.17

# За какво ще говорим

- Малко увод
- Java collections
- Свързан списък
- AVL дърво
- HashSet & HashMap

# Данни

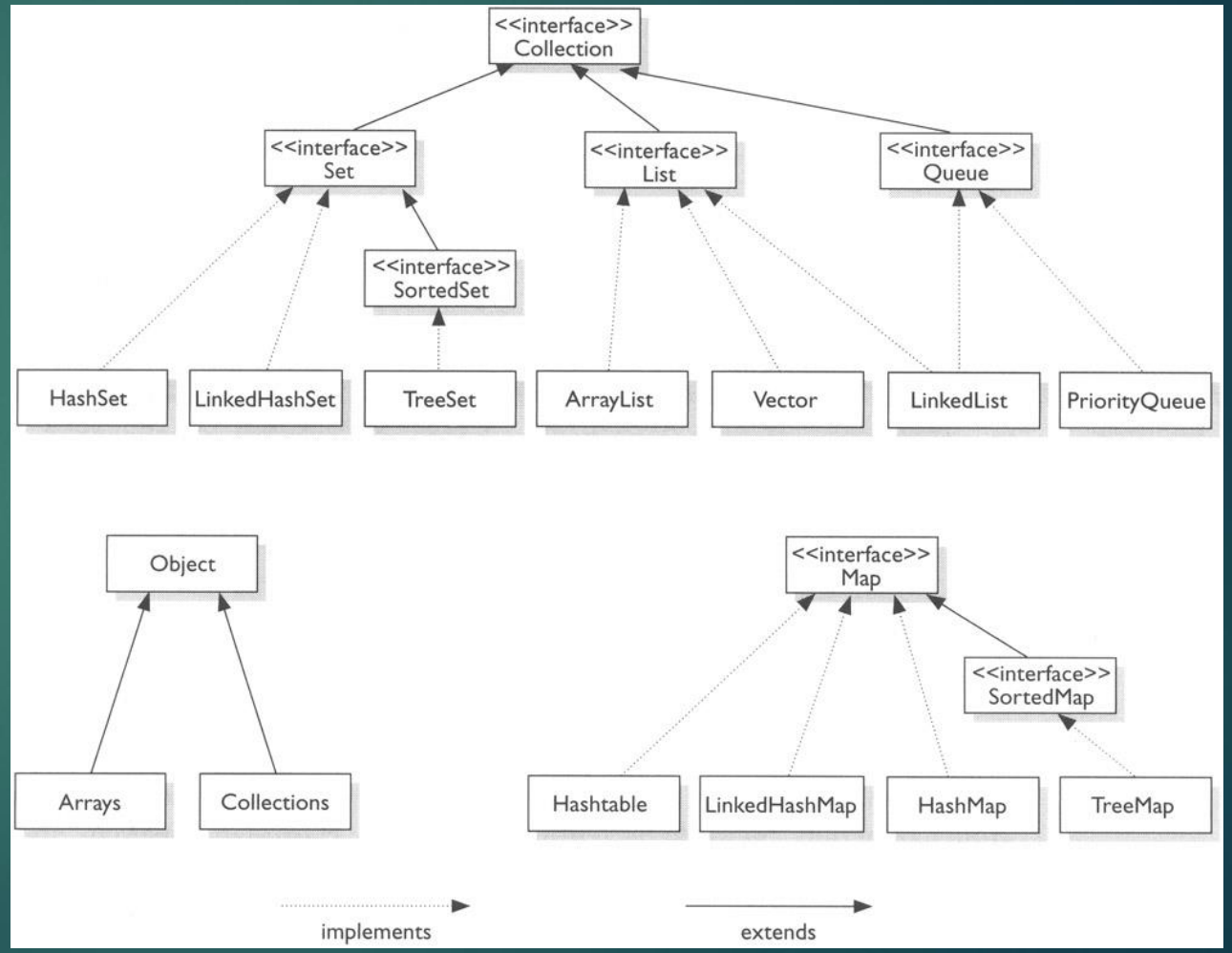
- Някакъв вид информация, която нашите програми получават като вход/създават и обработват
- Едно от основните неща, за които използваме компютрите, е да създаваме, съхраняваме и променяме данни
- Затова ни се налага да съхраняваме данните в компютъра по такъв начин, че да можем да работим с тях лесно и ефективно

# Абстрактен тип данни vs структура данни

- Абстрактен тип данни (АТД) – логическо описание
  - гледа се на типа данни от гледна точка на потребителя
  - *Какви* операции мога да извършвам с данните?
- Структура от данни – конкретно описание
  - как са подредени данните в паметта
  - *Как* се извършват операциите върху данните?
- Един АТД може да бъде реализиран чрез различни структури от данни
- За потребителя не трябва да е видимо коя точно
- Например списък може да бъде реализиран чрез структурите статичен/динамичен масив, едно- или двусвързан списък

# Структури данни в Java

- Колекция ≅ АТД + Структура данни
- The Java Collections framework:
  - Интерфейси т.е. абстрактни типове данни
  - Имплементации т.е. конкретни класове т.е. структури данни
  - Алгоритми
- import java.util.*;



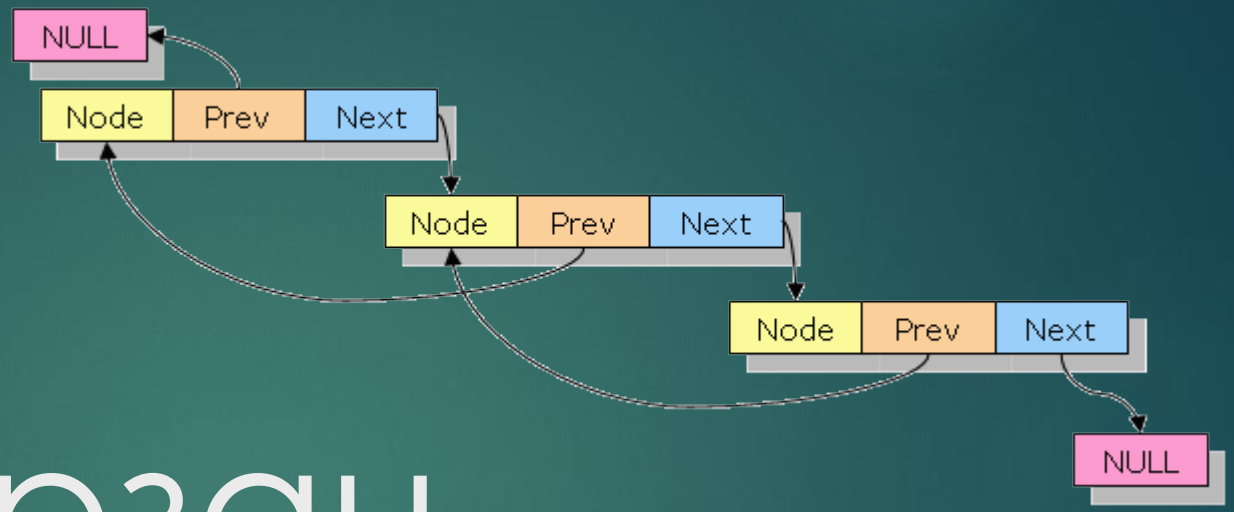
# Списък / List

# Абстрактен тип данни Списък

- boolean isEmpty()
- int size()
- int get(int index)
- void pushBack(int value)
- void pushBefore(int index, int value)
- int remove(int index)

# Имплементации на Списък

- Interface java.util.List;
- Implemented by:
  - ArrayList – статичен (обикновен) масив
  - Vector – динамичен масив
  - LinkedList – свързан списък
    - Едносвързан
    - Двусвързан – обикновено се ползва този
- LinkedList също имплементира интерфейса Queue
- Всяка от трите структури има плюсове и минуси – избираме според ситуацията




# Двусвързан списък

DOUBLY-LINKED LIST

# Интерфейсът List

- boolean isEmpty()
- int size()
- int get(int index)
- void pushBack(int value)
- void pushBefore(int index, int value)
- int remove(int index)

# ArrayList vs Vector vs LinkedList

| Операция | Масив | Вектор | Свързан списък |
|---|---|---|---|
| Достъп | O(1) | O(1) | O(n) |
| Търсене | O(n) | O(n) | O(n) |
| Добавяне/изтриване от позиция | O(n) | O(n) | O(1) |
| Добавяне/изтриване от края | O(1) | O(1) * | O(1) |

- * означава амортизирана сложност

# Java Generics

- Току що имплементирахме списък от цели числа
- Каква би била промяната в кода, ако искахме списък от обекти от тип Книга например?
- Доста малка!
- Затова използваме Java Generics

- Обикновено се стремим кода ни да е максимално generic (общ, независим от подробности)
- Подобна идея има и в други езици – C++ templates

# Generic интерфейсът на List

- boolean isEmpty()
- int size()
- Element get(int index)
- void pushBack(Element e)
- void pushBefore(int index, Element e)
- Element remove(int index)

# Стек и опашка

- Две от най-простите и известни структури данни са стек и опашка
- Те приличат много на свързания списък
- Просто имат по-малко от функционалността му
- Могат лесно да се имплементират чрез свързан списък
  - Вече не се налага да е двусвързан

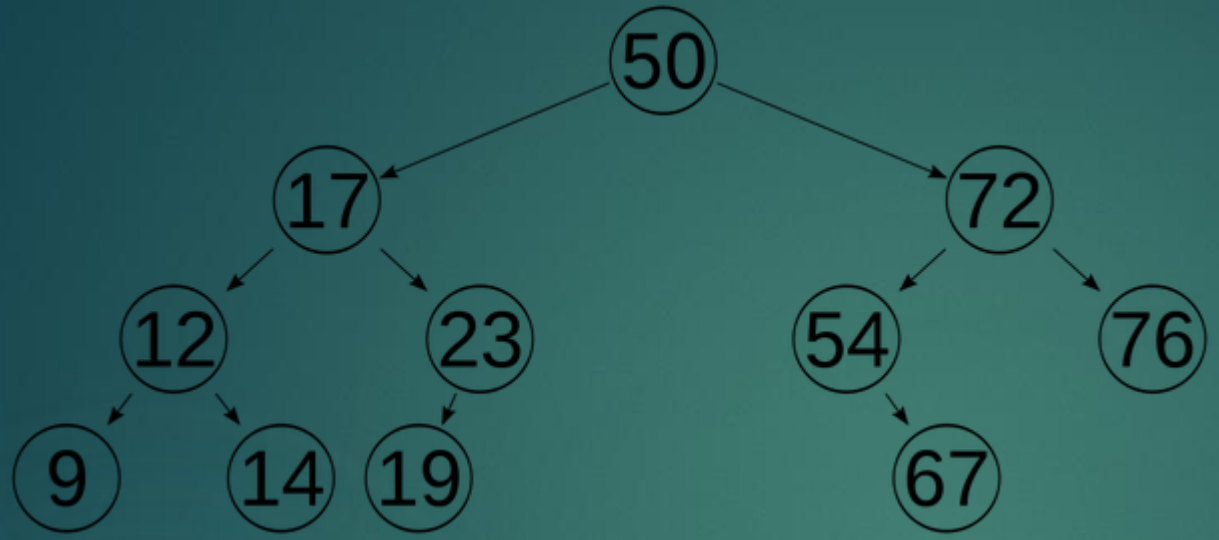
# Множество / Set

# Абстрактен тип данни Множество

- boolean isEmpty()
- int size()
- boolean contains(Element e)
- void add(Element e)
- void remove(Element e)

# Имплементация на Множество

- Interface java.util.Set;
- Implemented by:
  - HashSet
    - използва хеш таблица
    - ще го обсъдим малко по-късно
  - TreeSet
    - използва червено-черно дърво
    - ние ще покажем имплементация с AVL дърво (също ефективно, но по-просто)




# AVL дърво

ИЛИ НАКРАТКО

САМОБАЛАНСИРАЩО СЕ ДВОИЧНО ДЪРВО ЗА ТЪРСЕНЕ ☺

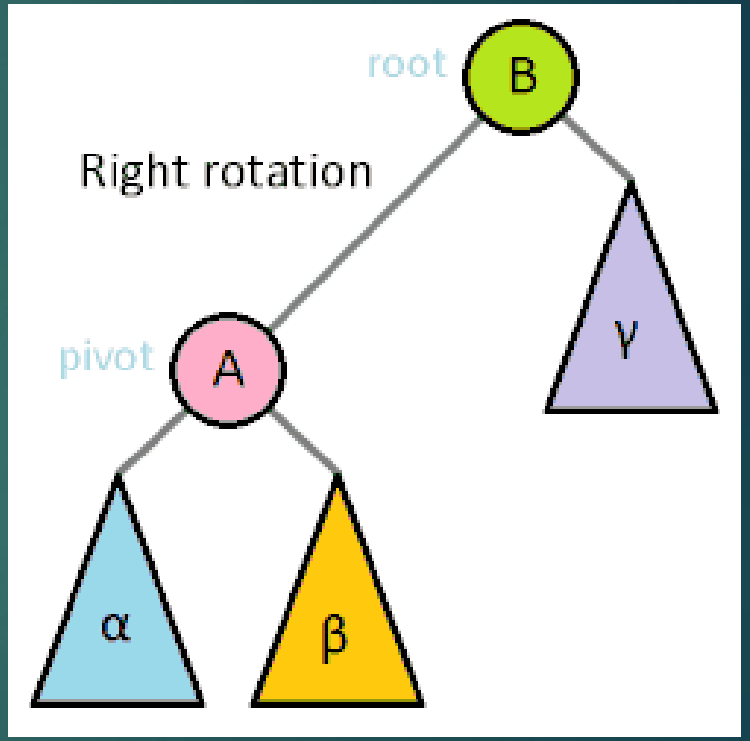
# Интерфейсът Set

- boolean isEmpty()
- int size()
- boolean contains(Element e)
- void add(Element e)
- void remove(Element e)

// да, вече е generic ☺

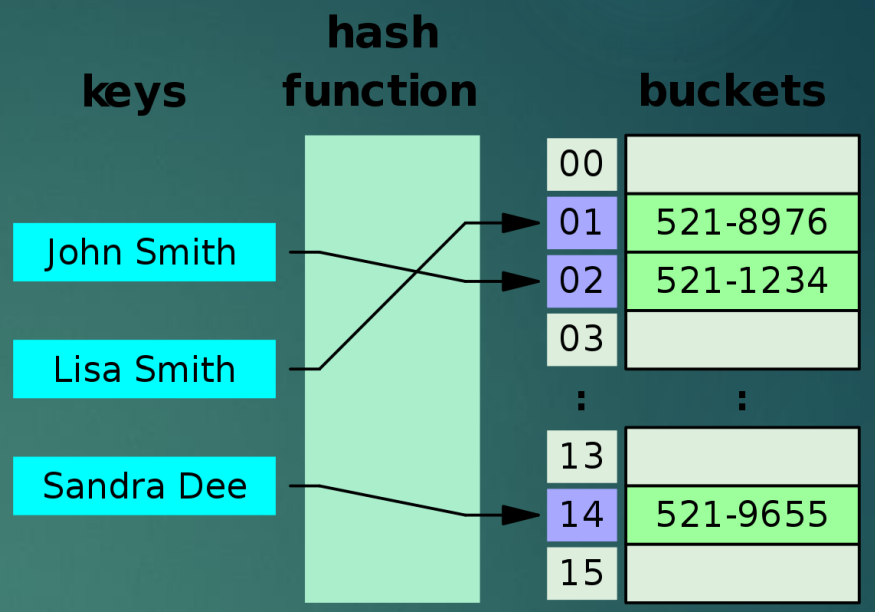
# Балансиране - чрез ротация

- Операциите стават все по-неефективни, когато дървото се „изражда“ в списък
- Затова го балансираме
- За всеки връх x на дървото искаме
  - (x.left.height – x.right.height) $\in \{-1, 0, 1\}$
- В противен случай правим ротация около x



# AVL дърво

| Операция | AVL дърво | Хеш таблица |
|---|---|---|
| Търсене | O(log n) | ? |
| Добавяне | O(log n) | ? |
| Изтриване | O(log n) | ? |




# Хеш таблица

HASH TABLE

# Хеш таблицата

- Един елемент в таблицата може да е:
  - Value ->  класа HashSet с интерфейс на Set
  - (Key, value) -> класа HashMap с интерфейс на Map

# Имплементация 2 на Множество

- Припомняме, че Множество е само интерфейс:
  - boolean isEmpty()
  - int size()
  - boolean contains(Element e)
  - void add(Element e)
  - void remove(Element e)
- Абстрактният тип данни (интерфейса) не ни ограничава с никаква конкретна структура от данни
- Преди малко видяхме имплементацията чрез структурата AVL дърво
- Сега ще покажем друга, използваща структурата хеш таблица

# Интерфейсът Set

- boolean isEmpty()
- int size()
- boolean contains(Element e)
- void add(Element e)
- void remove(Element e)

# Стратегии за справяне с колизии

- Отделно подреждане (separate chaining)
  - Чрез свързан списък
  - Чрез друга структура
    - HashMap от Java 8 например използва AVL дърво
- Отворена адресация (open addressing)
- Други?

# AVL дърво vs Хеш таблица

| Операция | AVL дърво | Хеш таблица |
|---|---|---|
| Търсене | O(log n) | O(1) |
| Добавяне | O(log n) | O(1) |
| Изтриване | O(log n) | O(1) |
| **Колизии?** | **Не** | **Да ☹** |