Java 8 in deep

By: Ahmad Yousif


Java

Agenda

- Java 8
- Why we should care!
- Collections & Maps
- Functional interface
- Lambda expression
- Introduction to Streaming api
- Let’s go in parallel


Krusty
Duff
RedBalls

Java 8

- Functional interface
- Lambda expression, let’s do some functional paradigm
- Method references
- Streaming api, let’s see some parallelism
- More details: http://www.oracle.com/technetwork/java/javase/8-whats-new-2157071.html

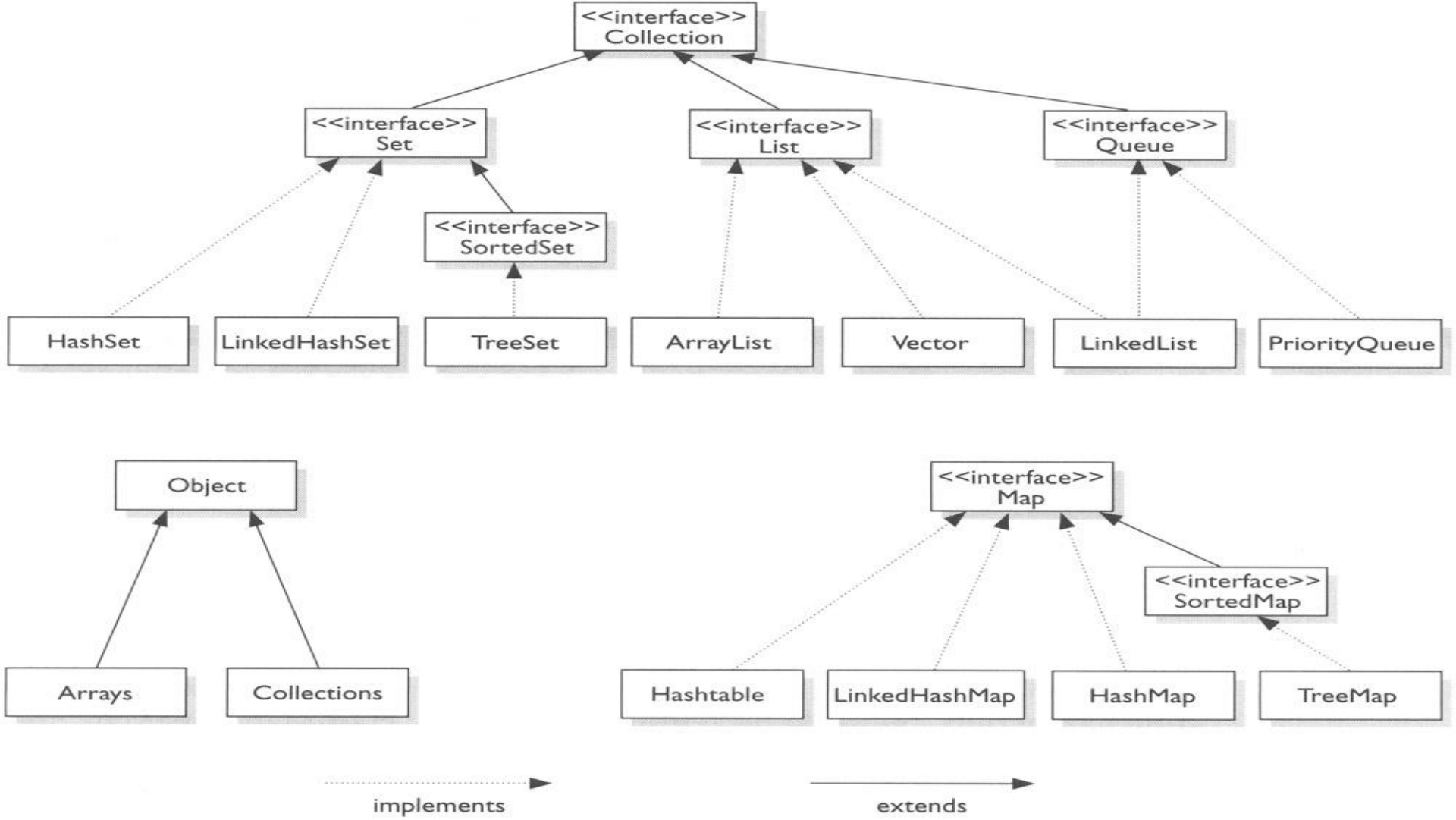
Why we should care!

- Welcome to complex problems
- Fast programming
- Less effort
- Welcome to functional paradigm
- I have cores, then i should use parallel computing


MTL

Collections & Maps

- Collections: parent of list, set, queue
    - List: duplication, insertion order, allows null values
    - Set: uniqueness, doesn't keep insertion order but LinkedHashSet does
    - Queue: fifo
- Maps: key-value style


<<interface>>
Collection
<<interface>>
Set
<<interface>>
List
<<interface>>
Queue
<<interface>>
SortedSet
HashSet
LinkedHashSet
TreeSet
ArrayList
Vector
LinkedList
PriorityQueue
Object
Arrays
Collections
<<interface>>
Map
<<interface>>
SortedMap
Hashtable
LinkedHashMap
HashMap
TreeMap
implements
extends

Functional interface

- It’s just an interface with one method
- What is the matter!
    - Useful with lambda expression
- Examples:
    - Predicate and test()
    - Function and apply()
    - Consumer and accept()
- You can declare your own by using @FunctionalInterface

Lambda expression

- New wave in java, sweet of programming
- I love functional programming, then you will love it
- It’s not new actually, rather than using anonymous interface you will use it
- Welcome new problems
- Less in code lines

Lambda expression

```
class  LambdaExp{
        Public static void main(String[] args){
                new Thread(new Runnable(
                 public void run(){
                        System.out.println("Hello World!");
                }
        )).start();
}
```

What about this!

```
class  LambdaExp{

        Public static void main(String[] args){

                new Thread(()->System.out.println("Hello World!")).start();

    }

}
```

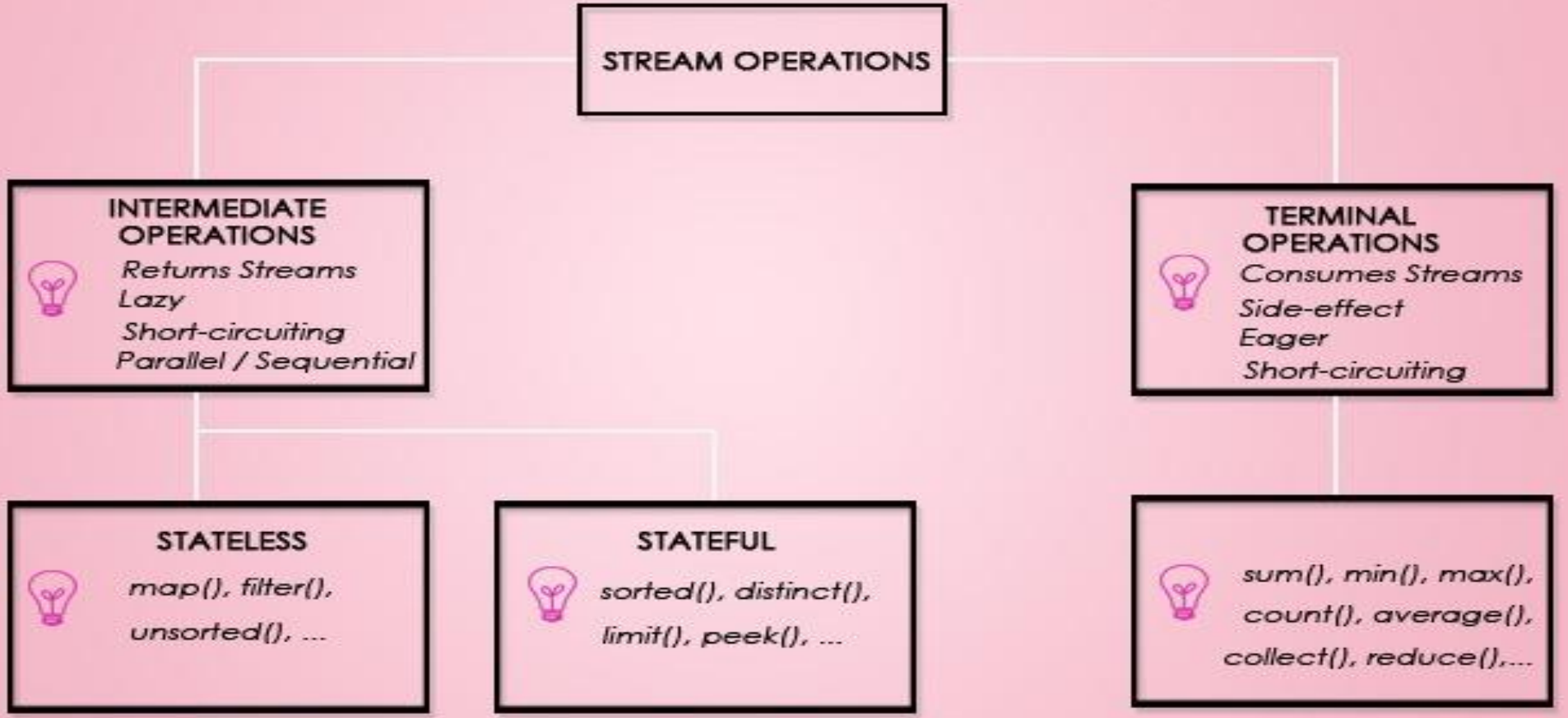
The picture can't be displayed.

Introduction to streaming api

- A collection is an in-memory data structure
- java Stream is a data structure that is computed on-demand
- Stream doesn’t hold data
- Moving the data from one pipe to another
- Can be sequential and parallel
- Under: java.util.stream

Stream example

```
Stream.of("d2", "a2", "b1", "b3", "c")
.map(s -> s.toUpperCase())
.filter(s -> s.startsWith("B"))
.forEach(s -> System.out.println(s));
//B1
//B3
```


STREAM OPERATIONS
INTERMEDIATE OPERATIONS
Returns Streams
Lazy
Short-circuiting
Parallel / Sequential
TERMINAL OPERATIONS
Consumes Streams
Side-effect
Eager
Short-circuiting
STATELESS
map(), filter(),
unsorted(), ...
STATEFUL
sorted(), distinct(),
limit(), peek(), ...
sum(), min(), max(),
count(), average(),
collect(), reduce(),...

Getting the stream

- Stream.of(1,2,3,4,5)
- Collection stream
- Steam.generate(() -> {return "abc";}) and Stream.iterate("abc", (i) -> i)
- Arrays.stream(new long[]{1,2,3,4})

Intermediate operations

- Stateless: doesn't require info. about other items
    - filter: do some condition on data
    - map: select and converting data type to another data type
- Stateful: require info. of other items
    - sorted
- flatMap(): getting the stream

Terminated operations

- reduce(): return Optional
- count(): return count number of items
- Matching: match one or all or none item based on condition
- foreach(): looping on item
- findfirst(): get the first item

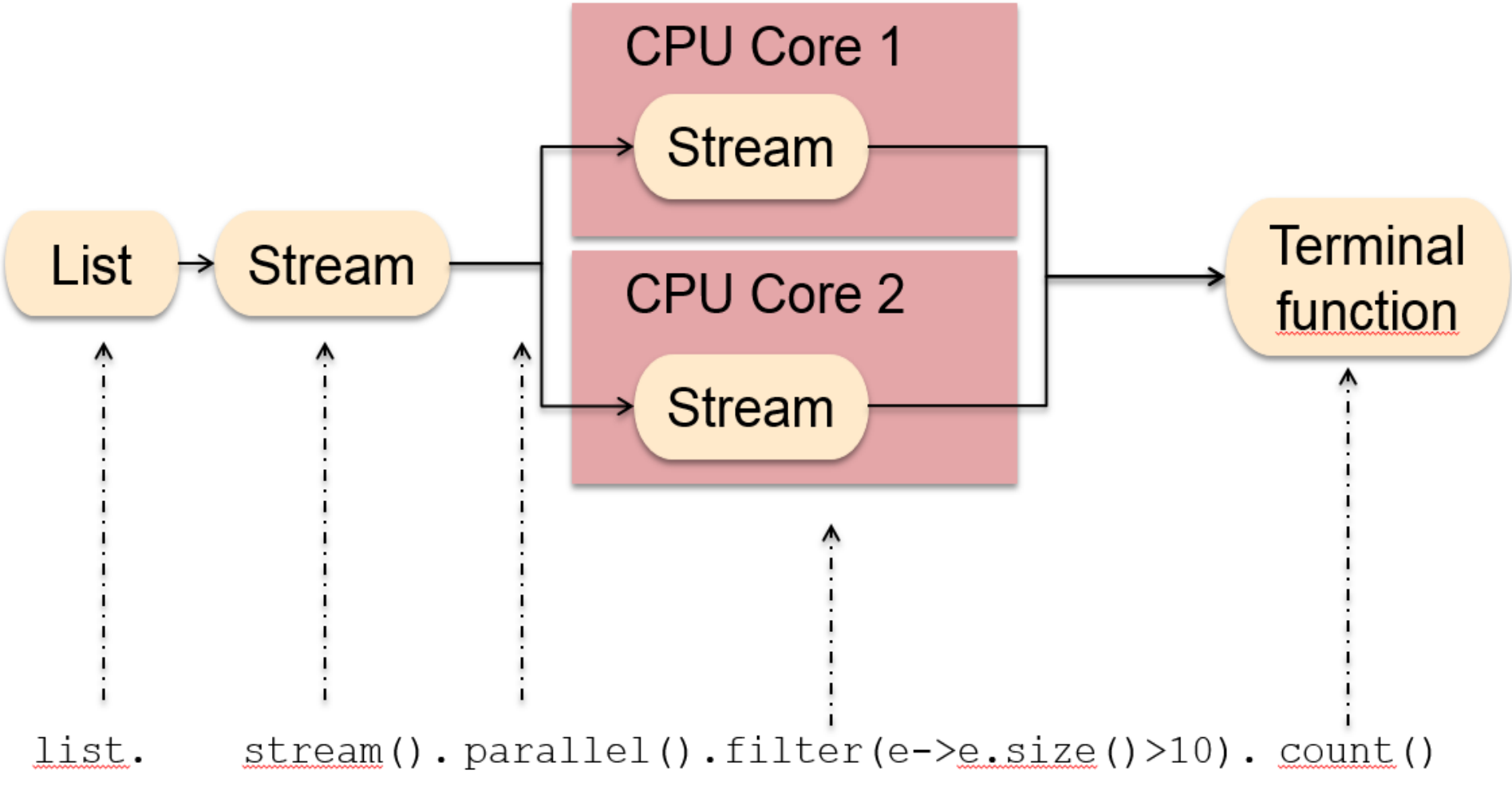
Lambdas refresh

- After Java 8:

```
List<Integer> ints = Arrays.asList(1, 2, 3, 4,
    5, 6);
ints.stream()
    .filter(i -> i % 2 == 0)
    .foreach(i -> System.out.println(i));
```

Let’s go in parallel

- You have more one core in cpu
- Log(n) can be Log(C), where c based on no. of core of processor
- How achieve this: parallelstream()


CPU Core 1
Stream
List
Stream
CPU Core 2
Stream
Terminal function
list. stream().parallel().filter(e->e.size()>10). count()


obrigado
Dank U
Merci
mahalo
Köszi
спасибо
Grazie
Thank you
mauruuru
Takk
Gracias
Dziękuję
Děkuju
danke
Kiitos