Cognixia®

# Working with Java 8

Cognixia®

# Functional Programming

# Lambda Expressions

- Why?
  - Alternate way of creating anonymous class instances
- Advantages
  - Easier creation of anonymous class instances
  - More readable anonymous class instances
- Ex: Runnable, FileFilter, Comparator interfaces and …
- The type of Lambda Expression : a Functional Interface
- Lambdas can be stored in a variable
- Can be used along with method references

# Lambda Expressions

- Collections processed using Lambda
- The forEach of Iterable<E>
  - Added in java 8 without breaking existing implementations
  - Default method : forEach
- The functional interface toolbox
  - Has several default methods

# The Functional Interfaces Utilities

- New Package
  - java.util.function
- Categories
  - Consumer
  - Predicate
  - Function
  - Supplier

Cognixia®

# Stream API

# Stream API

- Stream is
  - An object
    - on which operations are defined
    - Which does not hold data
    - Does not change data during computation
    - Processes data in a single pass
    - That processes data in parallel
    - With optimized algorithms

# Stream API

- Used for
  - Processing voluminous data
  - Processing smaller data too
- Processing mechanism
  - Parallel using multicore CPUs
  - Pipelined
- Stream is totally new
  - Collections work in the same old fashion.

# Backward Compatibility

- *Default methods* for interfaces
  - can still override it
  - don’t have to
  - Available through implementing class
  - forEach() in Iterable interface
- reuse interfaces
  - as a type of lambda expressions
  - Runnable, FileFilter, Comparator etc…
- Static methods
  - Available through interface

# Backward Compatibility

- Method Reference (::)
  - Used with
    - Instance & static methods
    - New keyword
- Loads of Functional Interfaces
  - The java.util.function package
  - The @FunctionalInterface annotation
- Streams
  - Works on existing collections
    - Sequential stream
    - Parallel stream

Cognixia®