Phase 1 : Java + Servlet API

Phase 2: Spring Framework

Phase 3: DevOps

Java - 8

IoT

Java-8 : Functional Programming

- Functional Interface
- default method
- static method
- Lambdas
- Streams
- Method references
- Optional
- Concurrent Support in Collection API
- DateTime API
- Nashorn Engine (JS engine)

Imperative style of programming

Classical style/Traditional style

pure OOPs

Focus how to perform operation

Object mutability : bugs

Declarative Style

Focus on result you want

Analogous SQL

Object Immutability

Functional Programming

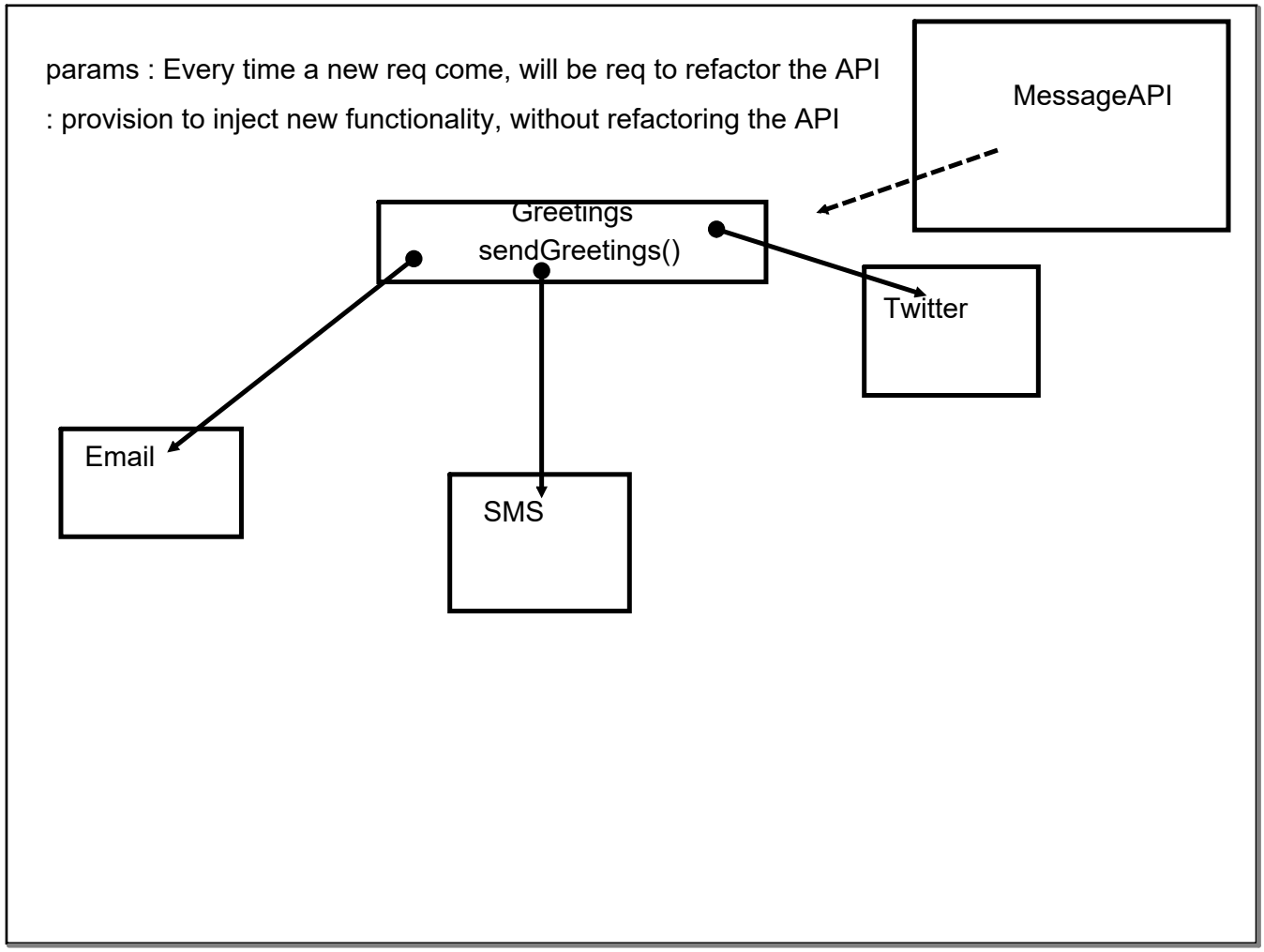
List of numbers

fetch unique number

Reverse domain naming convention

params : Every time a new req come, will be req to refactor the API

: provision to inject new functionality, without refactoring the API



Functional Programming : Functions(pure) are first class citizens

No Object Overheads

variable/instance/reference : object

reference = function

New datatype would not have been backward compatible

Expect a special datatype from JAVA : Function

Function twitter = ()

Extended the behavior of existing feature : interface

Syntax : Lambda

1. no access modifier : (not the part of any class)
2. no name (anonymous function)
3. no return type (can return values)
4. params : no param type
5. <param> -> {<definition>}

```
void fun(){
}
() -> {
}
```

```
void fun(String str1,String str2){
}
(str1,str2)->{
}
```

```
void fun (String str){
}
str -> {
}
```

```
void fun(String str){
      <single inst>
}
str-> <single inst>
```

```
void fun(String str){
----------
----------
}

str -> {
----------
----------
}
```

```
void add(int a, int b){
      return a+b;
}
(a,b)-> a+b;  // return is by default associated
(a,b) -> {
      return a+b;
}
```

Functional Interface

Contains only 1 abstract method, any number of default and static

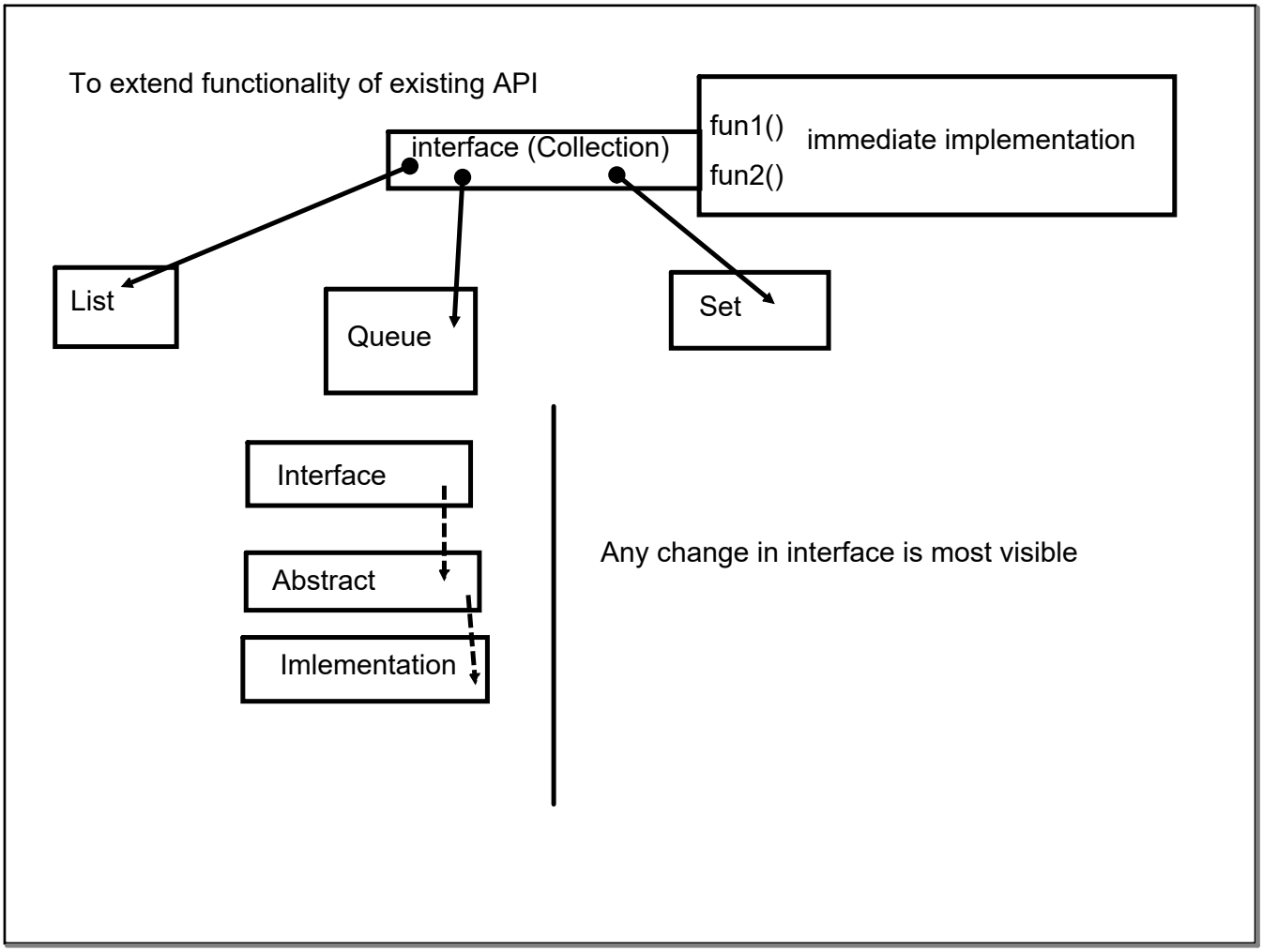
Only Functional Interfaces can refer to lambda expression

Signature of Lambda expression must match with the only abstract method of FI

Interface :

Define function inside an interface.

Interface can have functions with definitions as well


To extend functionality of existing API
interface (Collection)
fun1()
fun2()
immediate implementation
List
Queue
Set
Interface
Abstract
Imlementation
Any change in interface is most visible

Existing feature :

#Functional Interface

Comparable

Comparator

Runnable

=> Specialized Libraries of Functional Interface

=> Streams

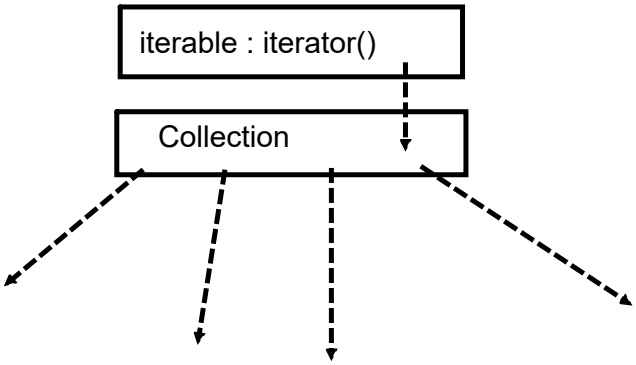
Lambdas with Local Variable

1. Lambdas have an access over local variables and instance of enclosing scope
2. Effectively final
3. Not Allowed to use the same local variable name as param or redeclaration inside body

No restriction on instance variable

=> Easier to perform the concurrent operation : immutability

Collection of string : traverse through collection and display all strings in collection



Functional API : Bunch of functional interface:

few prototypes have been identified with common usage

java.util.function

Consumer: BiConsumer

void accept(<>) : Consume the data

Predicate : BiPredicate

boolean test(<>) : Add some condition and revert back

Function : BiFunction, UnaryOperation, BinaryOperator

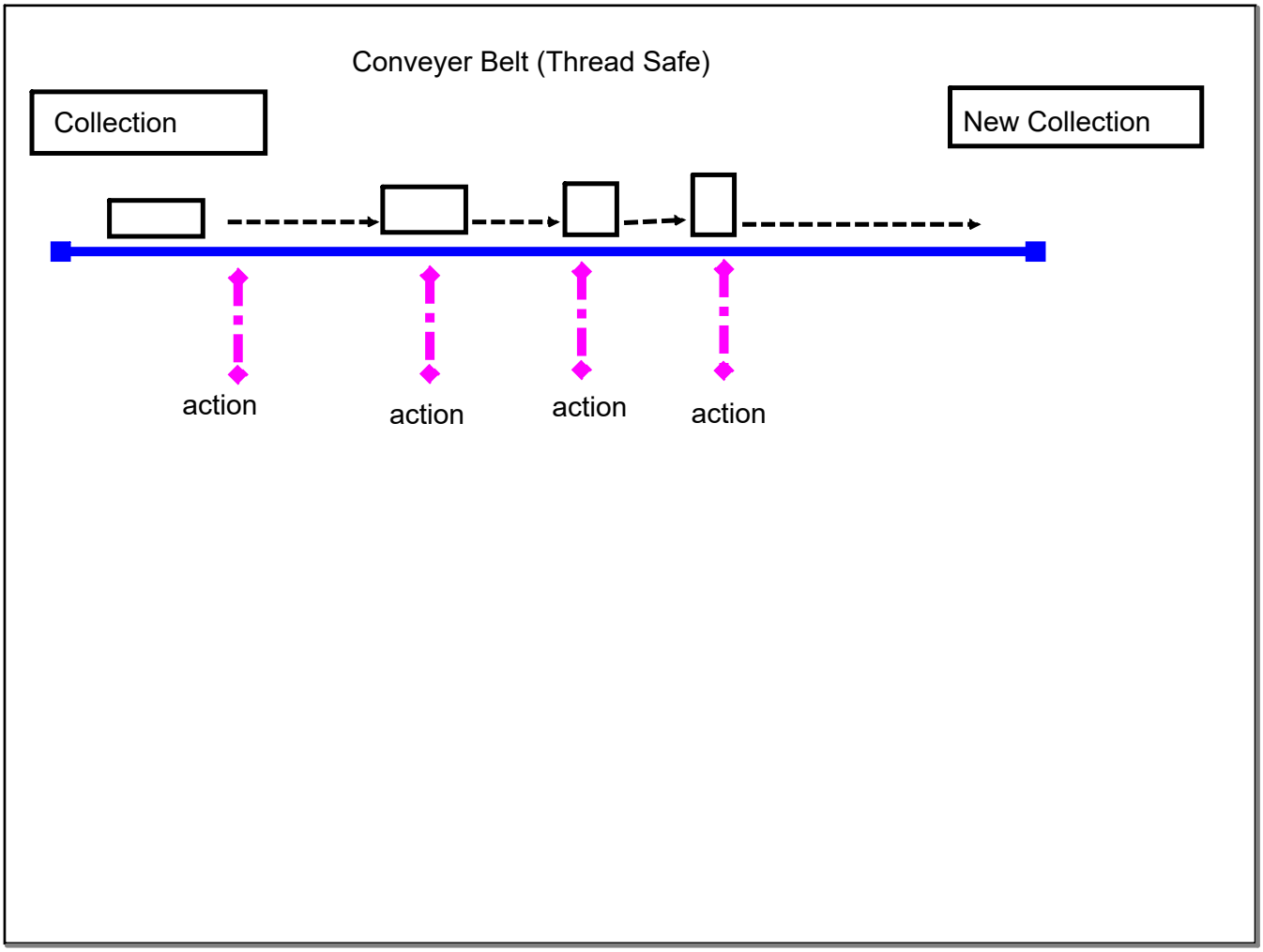
<> apply(<>) : Transformation

Supplier

<> get() :

Streams : Pure Functional

- Perform operations on collections or I/O resource :
- Safe
  - Immutability : Thread safe
- Efficient Way
  - Not a data structure: not going to store any data
  - Lazy processing model
- Parallel Stream : Parallel operation easily without spawning the thread


Conveyer Belt (Thread Safe)
Collection
New Collection
action
action
action
action

SBA1 : use-case

SBA2 : use-case

SBA3 :

End-to-End

1. continous process
2. Milestone
3. walk-through (Friday)
4. Group based : group evaluation + individual eval

Every Stream must have a terminal activity

Else : Stream will not initiate