Phase 1 : Java

Phase 2 : Spring Framework

Phase 3 : DevOps Tools

Java - 8 : IoT

Lambda Expression

base64 API

Streams

Functional Interface

default

method reference

Optional

DateTime API

Concurrent API enhanced

Nashorn Engine (JS engine)

Functional Programming

functions as first class citizens + OOPs

Imperative style of programming

Classical/Traditional

# Focus : how to perform operation

# write steps on how to achieve an objective

# Object mutability

Declarative style of programming

# Focus : what is the result

# Object immutability

# SQL style

Collection of numbers

fetch the unique numbers

wf.com

com.wf

<reverse domain of org><training>

Greeting API:

convey Message : console

: email

: sms

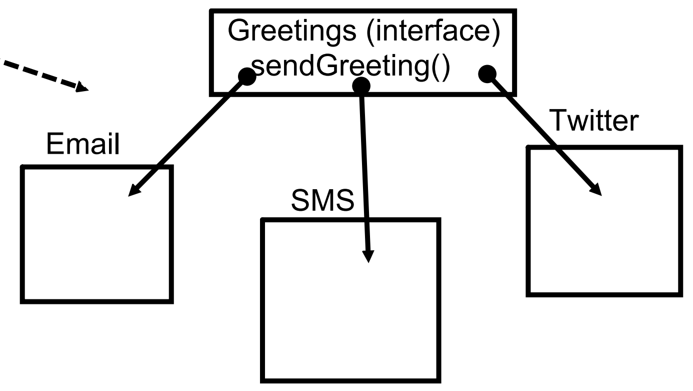
=> twitter

=> network : URL

Refactor the API not recommended:

inject new functionality : inheritance

Loose Coupling



Declarative:

Inject only functionality (pure function), not wrapped inside an object

Java should expose a datatype : Function

New Datatype : would not be backward compatible

interface : Function datatype

Syntax :

1. not have any access modifier
2. Anonymous function (no name)
3. return type is not mentioned
4. no param types
5. <praram name> -> {<definition>}

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

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

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

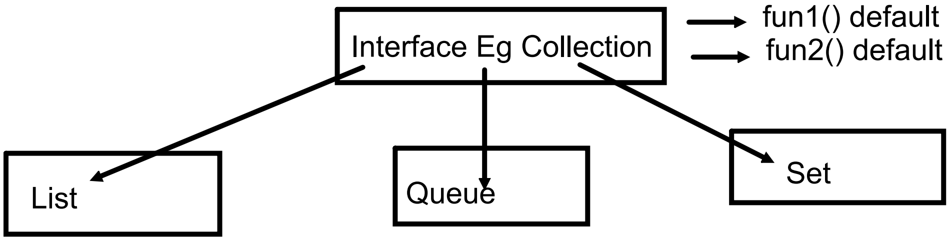
```
void fun(String str){
      ..only single instruction
}
str-> <single instruction>
```

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

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

interface :

1. default functions : interfaces can have functions with definition



Functional Interface :

An interface containing only a single abstract method

it may have multiple default and static method

Lambdas/Method Reference can be assigned to only functional interface reference

Lambda Expression/ Method Reference signature must match with the only abstract method of FI

An reference of functional interface can refer to any method as long as its signature matches with the only abstract method (other than lambdas also)

More Practical Usage....

Streams

Specialized lib/api

Existing interface

# Runnable

# Comparator

# Comparable

Lambdas with local variables

# Effectively final : Local variable declared outside the Lambdas are effectively final inside the Lambda expression

# Not allowed to use the same local variable name as param or inside the lambda body

# Not restriction for instance variable

-> Easy to perform concurrency operations

-> immutability

A Special Library of Functional Interfaces

# Common prototypes are exposed

java.util.function.*

Consumer : BiConsumer

void accept(<>) : Consume the data

Predicate : BiPredicate

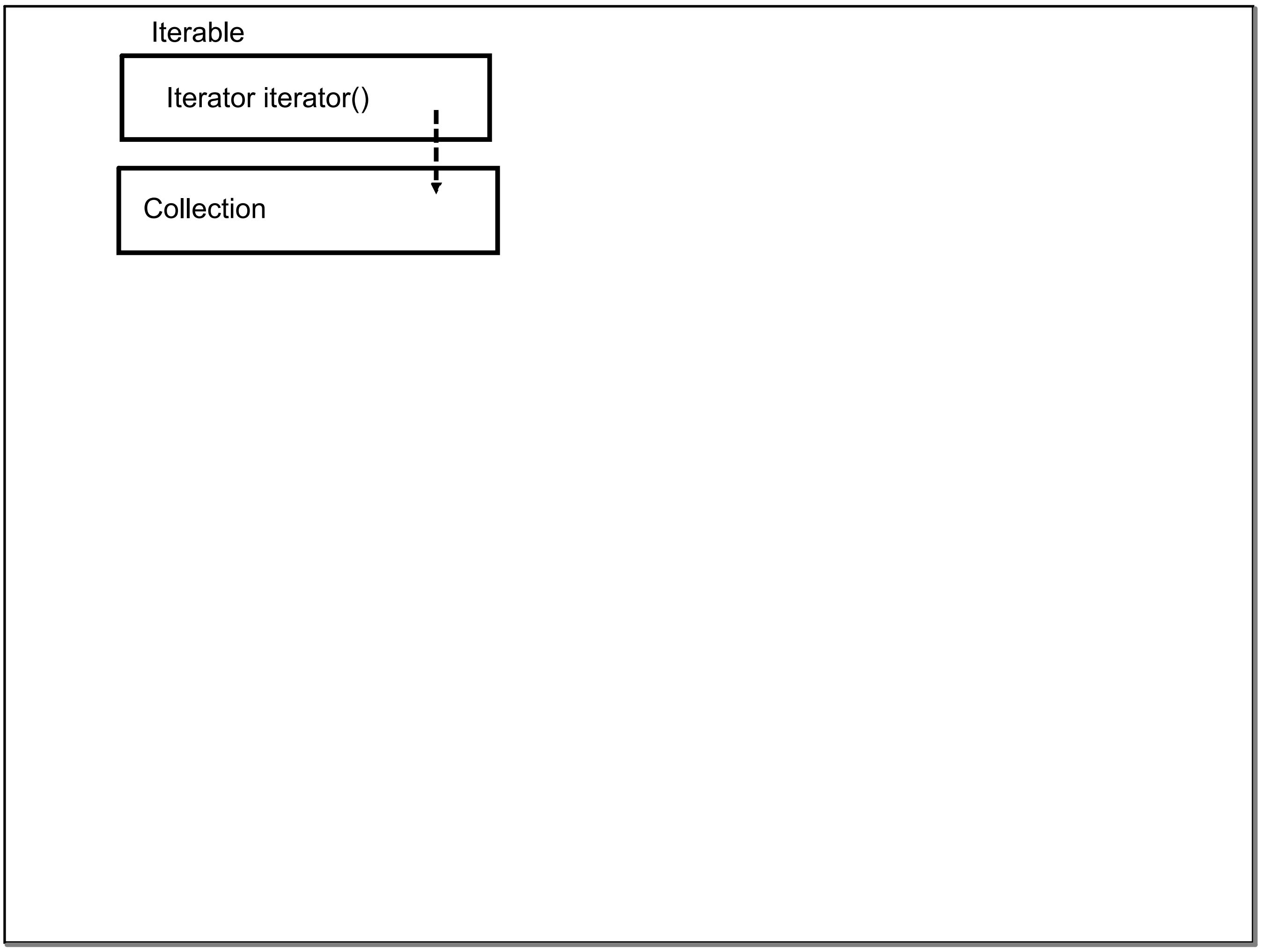
boolean test (<>) : Use data to check some condition

Function : BiFunction, UnaryOperator, BinaryOperator()

<> apply(<>) : Transformation

Supplier:

<> get() : Generate some data and return data back


Iterable
Iterator iterator()
Collection

Stream : Sequence of elements created out of collections / IO resource

Add a stream of activity

Stream : Safe and Efficient way

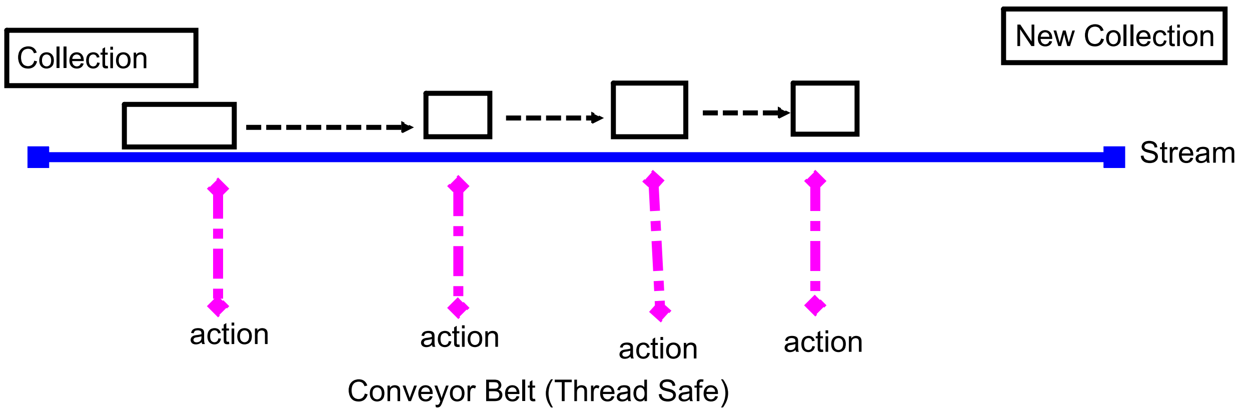
Safe :

immutability : Thread safe implementation : Concurrency

Sequential/ Parallel

Efficient :

Not a Data Structure : not going to store any data : Lazy processing model



Phase1 : SBA1 :

Phase2 : SBA2

Phase3 : SBA3

Use case :

End-to-end : Milestone (weekly)

Team based implementation