Functional Interfaces

Mr. Pankaj Jagasia

Functional Interface

- A Functional Interface is a POJI which contains 1 and only 1 abstract method.
- Was introduced to support functional programming through Lambda Expressions
- A Functional Interface may contain any number of default and static methods
- @FunctionalInterface Annotation introduced to specify the interface is *Functional* and should not contain more than 1 methods.
- If Parent interface is Functional then a Blank Child Interface can be *Functional*
- If Parent interface and Child have the same signature method and both contain only 1 method each then the interface can be *Functional*
- Child Interfaces containing even a single method cannot be marked as *Functional*
- *Functional Interfaces* can be inherited into Child Interfaces as long as the Child Interface containing methods is not marked as *Functional*

Predefined Functional Interfaces

Single Parameter FunctionalInterface

- Predicate
- Function
- Consumer
- Supplier

Two Parameter FunctionalInterface

- BiPredicate
- BiFunction
- BiConsumer

Accepting and Returning same Parameter same as Function
FunctionalInterface

- UnaryOperator
- BinaryOperator
- Primitive type FunctionalInterface
- IntUnaryOperator
- IntBinaryOperator

Primitive type FunctionalInterface

- IntPredicate
- IntFunction
- IntConsumer

Predicate

- Used for testing logical condition returning true or false
- “test” is the only abstract method in this interface
  - public boolean test(T a);
- “and” to check 2 predicates satisfying the condition
- “or” to check 2 predicates satisfying any 1 condition
- “negate” or “not” to check negative condition like the != operator

Predicate Joining

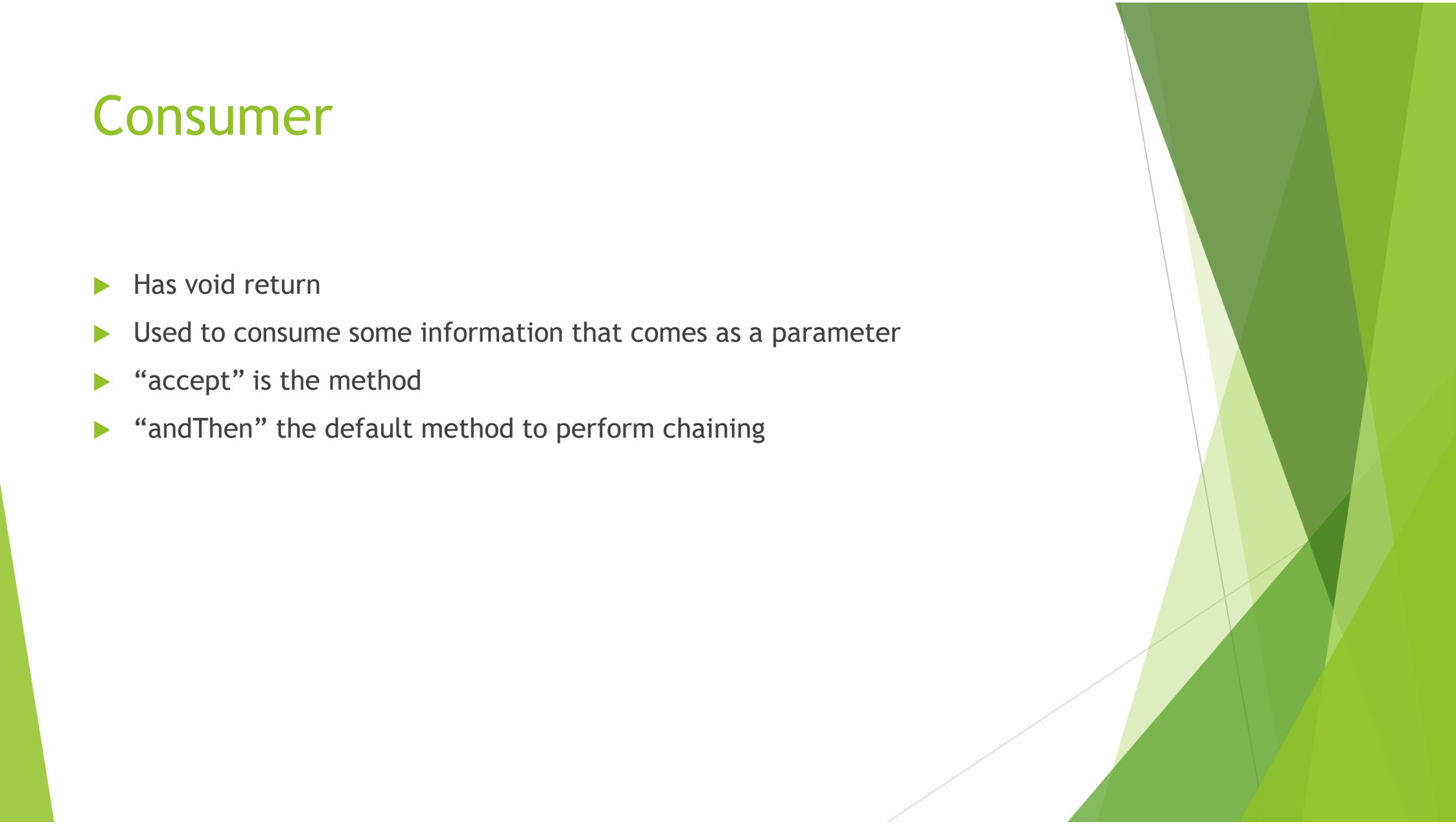
- Using negate,and,or,not one can join multiple Predicates to test various conditions
- Eg
  - Predicate<Integer> condition1 = (I)->I>100;
  - Predicate<Integer> condition2=(I)->I%2==0;
  - Condition1.and(condition2);

Function<P,R>

- P represents the argument and R the return type
- So a functional interface that can take an argument and return any value
- “apply” the abstract method in it.
- “andThen” the default method to use Function chaining
  - F1.andThen(f2).apply(x) In this case F1 will be executed 1st then F2
- “compose” the default method to use Function chaining
  - F1.compose(f2).apply(x) In this case **F2** will be executed 1st then F1

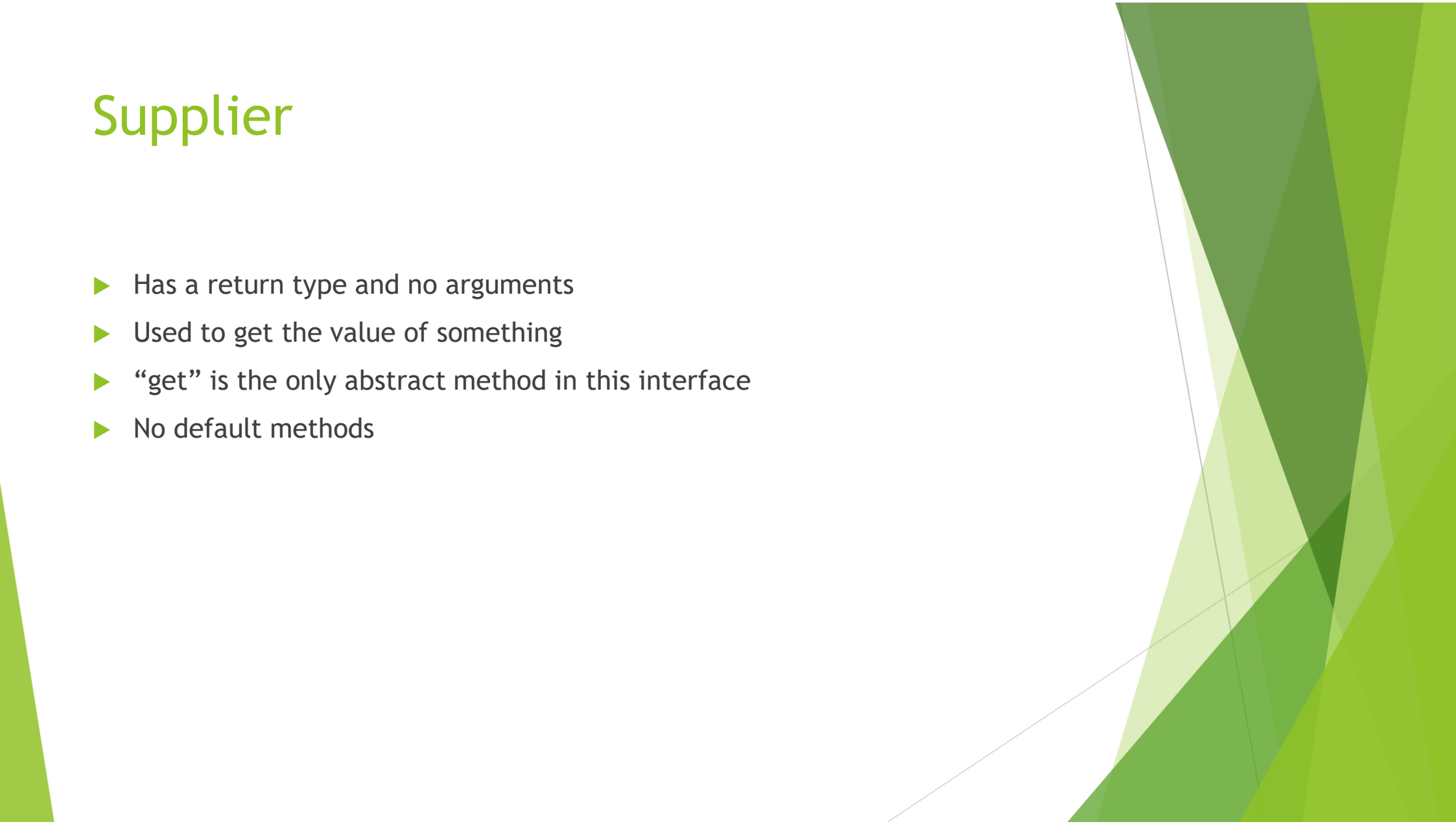
Consumer

- Has void return
- Used to consume some information that comes as a parameter
- “accept” is the method
- “andThen” the default method to perform chaining



Supplier

- Has a return type and no arguments
- Used to get the value of something
- “get” is the only abstract method in this interface
- No default methods



Comparing Predicate and Function

Predicate

- Used for checking conditions
- Takes a single argument
- “test” method
- Returns boolean

Function

- Used for checking conditions
- Takes 2 arguments 1 for input and 1 for return
- “apply” method
- Returns any type