Java 8

Interface

TestLeaf
Always Ahead

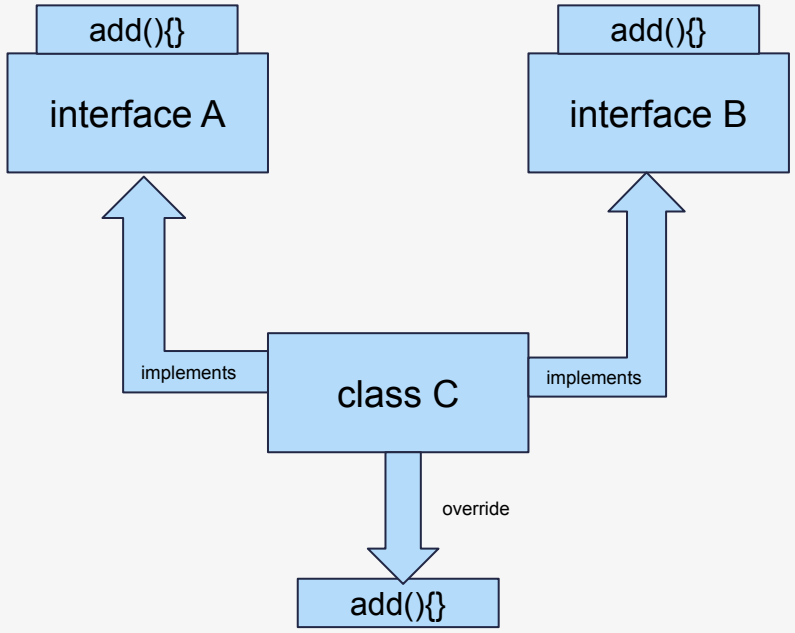
Before Java 8

- We can only declare a method in Interface
- By default it is “public abstract”

From Java 8

- We can only declare and define a method in Interface
- By this they achieved backward compatibility
- Interface can now take static functions

Diamond Problem in Java 8 Interface



- Two interface having same function
- Class extend both the interface
- Ambiguity arises here
- To overcome this we need to override the methods in the implemented class

default method in interface

- To declare a method inside the interface default keyword is used

```
interface A {
        default void add(){
                system.out.println(“added”)
        }
}
```

static method in interface

- From Java 8 we can also declare static method inside the interface call them directly.

```
interface A {
        static void add(){
                system.out.println(“added”)
        }
}
```

For loop

Traditional For loop

```
List<Integer> lst = Arrays.asList(1,2,3,4,5,6);
for(int i = 0; i < lst.size(); i++){
        System.out.println(lst.get(i));
}
```

Enhanced For loop

```
List<Integer> lst = Arrays.asList(1,2,3,4,5,6);
for(Integer i : lst){
        System.out.println(i);
}
```

Foreach loop

- It is default method in iterable interface
- It pass each element and perform actions on each element

```
List<Integer> lst = Arrays.asList(1,2,3,4,5,6);
lst.forEach(i -> System.out.println(i));
```

Lambda

TestLeaf
Always Ahead

Code in oops

- Everything is an object
- All code blocks are associated with classes and objects

Functions as values

- Inline values :
  - String name = “foo”;
  - double pi = 3.14;
- aBlockOfCode = {
  …
  }

Why Lambda

- Enables Functional programming
- Parallel programming

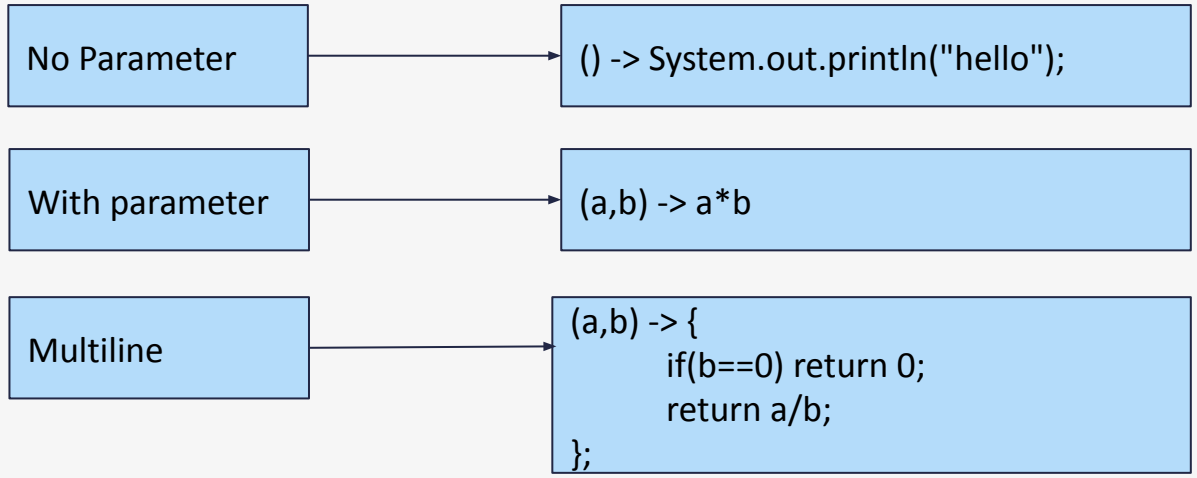
Syntax

```
parameter -> expression body
```

Important characteristics of a lambda expression

- **Optional type declaration** − No need to declare the type of a parameter. The compiler can inference the same from the value of the parameter.
- **Optional parentheses around parameter** − No need to declare a single parameter in parenthesis. For multiple parameters, parentheses are required.
- **Optional curly braces** − No need to use curly braces in expression body if the body contains a single statement.
- **Optional return keyword** − The compiler automatically returns the value if the body has a single expression to return the value. Curly braces are required to indicate that expression returns a value.

Example:


No Parameter
() -> System.out.println("hello");
With parameter
(a,b) -> a*b
Multiline
(a,b) -> {
if(b==0) return 0;
return a/b;
};

Inbuilt Interfaces

- Java 8 has some inbuilt interfaces to address some of these common scenarios
- Package : java.util.functions
- Some of the commonly user interfaces are
  - Predicate
    - Takes input argument and return boolean value
  - Consumer
    - Takes input argument and return nothing
  - Supplier
    - Takes nothing and return a object

Method Reference

- Method reference is used to refer method of functional interface.
- It is compact and easy form of lambda expression.
- Each time when you are using lambda expression to just referring a method, you can replace your lambda expression with method reference

Types of Method References

- Reference to a static method.

```
ContainingClass::staticMethodName
```

- Reference to an instance method.

```
containingObject::instanceMethodName
```

- Reference to a constructor.

```
ClassName::new
```