Lambda Expressions

Mr. Pankaj Jagasia

What is a Lambda Expression

- It is in anonymous function that does not have
  - A name
  - A return type
  - A modifier
- Was already available with other languages such as LISP, C, C++, Scala, C#, Ruby etc

- Benefits
  - Enable functional programming
  - To write more readable, maintainable & reduced code
  - Using APIs easily and efficiently
  - To enable parallel processing
- Easy to Use

• Standard code	• Lambda Expression	• Simplified Lambda Expression

int add(int num1, int num2)
{
return num1 + num2;
}
	(int num1, int num2) ->{
	return num1 + num2;
	}
		(num1,num2) ->
		num1 + num2;

Concise coding in Java using Lambda

- In Lambda the types of arguments can be automatically be identified by the Compiler this is called as Type Inference for eg.

  (num1,num2) -> return num1+num2; [assuming they are int]

- If Lambda Expression contains only 1 statement then
  - the { } braces are optional.
  - The return statement is optional [(int num1, int num2) -> num1+num2;]
  - The return statement and argument type is optional if it contains only 1 statement eg [(~~String~~ objString) -> ~~return~~ s.length();]
    i.e. (objString) -> s.length();
  - If the function is taking only one argument you can skip the ( ) eg.
    objString -> s.length();

Anonymous Classes vs Lambda Expression

- It's a class without a name
- Can be sub-classed from Abstract or Concrete classes
- Can implement interfaces that have n number of methods
- Can contain instance variables
- this refers to the inner class object
- Separate class generated by the compiler
- Memory will be dynamically allocated at runtime

- It's a method without a name
- Can be used with Functional Interface only.
- Lambda can be used with interfaces having 1 and only 1 method
- Cannot contain instance variables
- **this** refers to the containing object
- No additional class generated
- Memory treated as a part of the code.