**Functional Interface:** Exactly one abstract method. provides way to invoke Lambda Expression

**List**

Insertion order is preserved

duplicated allowed

ArrayList, LinkedList (Vector -> stack)

**Set**

duplicates not allowed (adding duplicates, no errors, but only one object is maintained)

order is not preserved

HashSet, TreeSet (useful for sorting)

**Map**

key, value pair

Order not preserved for Hash

**Comparator**

int compare(Object obj1, Object obj2)

obj1 < obj2 --> -ve obj1 come before obj2 in sorting

obj1 = obj2 = 0

obj1 > obj2 --> +ve obj1 come after obj2 in sorting

**All Anonymous classes can't be converted to lambda expressions:**

Can we write Lambda?

Anonymous inner class that extends concrete class. (NO)

Anonymous inner class that extends abstract class. (NO)

Anonymous inner class that implements an interface that contains more than one interface. (NO)

Anonymous inner class that implements an interface that contains Only one interface. (YES)

**this** in anonymous and lambda

Inside anonymous inner class we can declare instance variables. inside lambda we can not only

local variables. and in anonymous inner class this always refers to instance of inner class.

in lambda expression this refers to outer side class members.

**Difference between Anonymous Inner class and Lambda Expressions**

|  |  |
| --- | --- |
| **Anonymous Inner class** | **Lambda Expression** |
| It is a class without name | It is a function without name |
| I can extend Abstract class and concrete class | Lambda can’t |
| Can implement interface with multiple methods | can implement interface with only one method |
| Inside AIC we can declare instance variables | We can’t declare instance variables, what ever variables we declare are local variables |
| AIC can be instantiated | Cannot be instantiated |
| in AIC, this always refers to current Anonymous Inner class object but not outer class object | this always refer to outer class object, ie enclosing class object |
| A separate class file will be created when compiled | No separate class. |
| Memory will be allocated on demand whenever, we are creating object | Lambda expression will reside in permanent memory of JVM(Method area) |

Any local variable access from Lambda expression is final by default, whether the variable declared as final or not.

**Advantages of Lambda expressions:**

1. Reduce length of code, more readable code.
2. We can enable functional programming in Java
3. Resolve complexity of Inner class until some extent
4. We can handle procedures/functions just like values/arguments
5. Easier to use updated APIs and Libraries
6. Enable support for parallel processing.

**Default Methods**

**Every method in Interface:** public, abstract (till 1.7)

**Every Variable:** public, static, final

From 1.8 we can method with implementation with default keyword, then what is the difference between abstract class and interface?

|  |  |
| --- | --- |
| **Interface with default method** | **Abstract class** |
| Inside interface every variable is public, static, final. We cannot declare instance variables | We can declare instance variables required for child class |
| Interface never talks about state of object | Abstract class can talk about state of object |
| No constructors | Can have constructors |
| No static and instance blocks | Static blocks, instance blocks allowed |
| Functional interface with default methods can refer Lambda expression | Abstract class can’t refer lambda expressions |
| Inside interface we can’t override object class methods | Inside abstract class we can override object class methods. |

**Static Methods**

Any class can call these static methods, doesn’t need to implement this interface. Just to define general utility method. Only way to call this method is InterfaceName.methodName();

By default, method is not available to the implementation class, always use InterfaceName and method name to invoke.

Interface static methods wrt Overriding: overriding concept not applicable for interface static method. We can use same method name in class as static method, but not overriding.

Interface have method: public static void m1() { }

Class can have any one of these method signatures:

Public static void m1() {}

Public void m1() {}

Private static void m1() {}

**Functional Interfaces (java.util.function)**

1. Predicate: perform some conditional check and return true or false.
2. Function
3. Consumer
4. Supplier

Interface Predicate<T>{

boolean test(T t);

}

**Predicate Joining**

1. negate()
2. and()
3. or()