**Lambda**

A lambda is function.

A function is a computation that takes parameters and returns a value.

Until java 8, function could only be implemented using methods.

A lambda enables functions to be passed around or stored like data.

A lambda expression is something like a method without a name.

**Using an anonymous inner class the code looks like this:**

Comparator<Integer> comp = **new** Comparator<Integer>() {

@Override

**public** **int** compare(Integer i1, Integer i2) {

**return** i1 - i2;

}

};

**Using lambda above code looks like below:**

Comparator<Integer> comp = (Integer i1, Integer i2) -> {

**return** i1 - i2;

};

When writing lambda, there is no need to define argument types. Above, comp variable represents a behavior instead of object and this behavior can be passed as an argument to a method.

Above written code can be reduced.

Comparator<Integer> comp = (i1, i2) -> {

**return** i1 - i2;

};

If there is only one statement, then neither return keyword nor curly braces required.

Above written code can be further reduced.

Comparator<Integer> comp = (i1, i2) -> i1 - i2;

**Example of lambda**

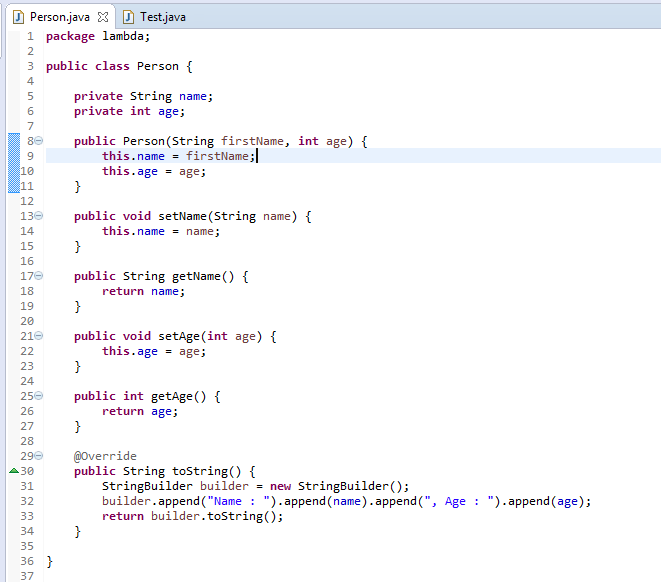


Figure 1.0

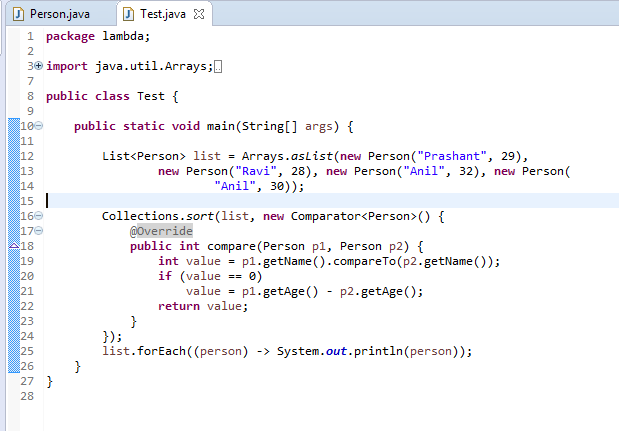


Figure 1.1

In figure 1.1, anonymous class is created for Comparator and its compare method is defined. Wherever this comparator is required, it would be defined this way everywhere. We can see that there is lots of boiler plate code, which we need to write again and again.

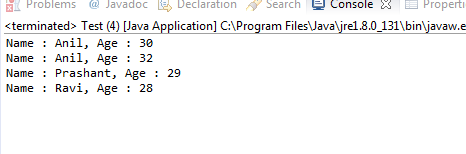


Figure 1.2

**Using lambda**

Behaviour is stored in a variable, which is passed as an argument to sort method.

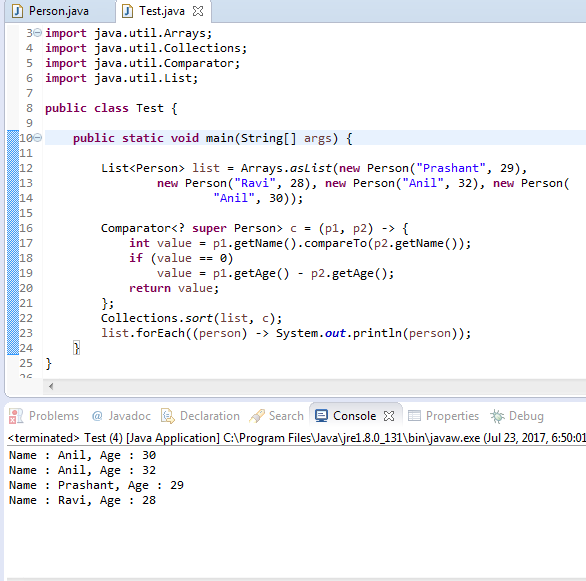


Figure 1.3

We can also pass the behavior without storing it in a variable.

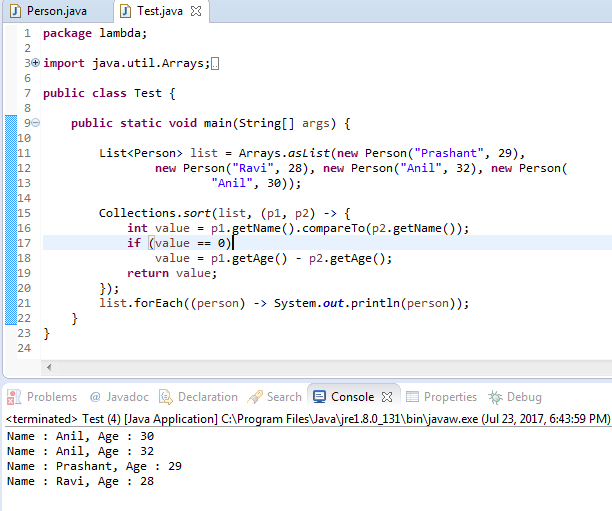


Figure 1.4

Reduced form of lambda with respect to the code in figure 1.4

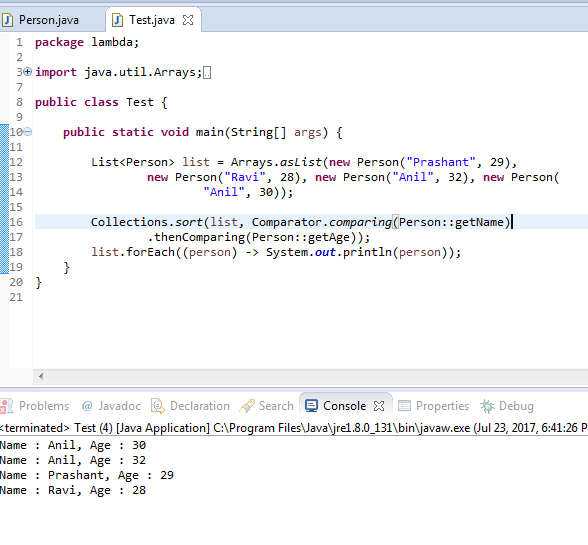
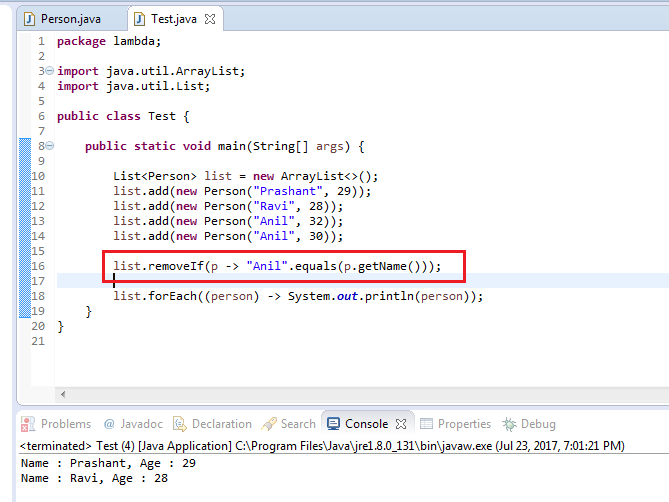


Figure 1.5

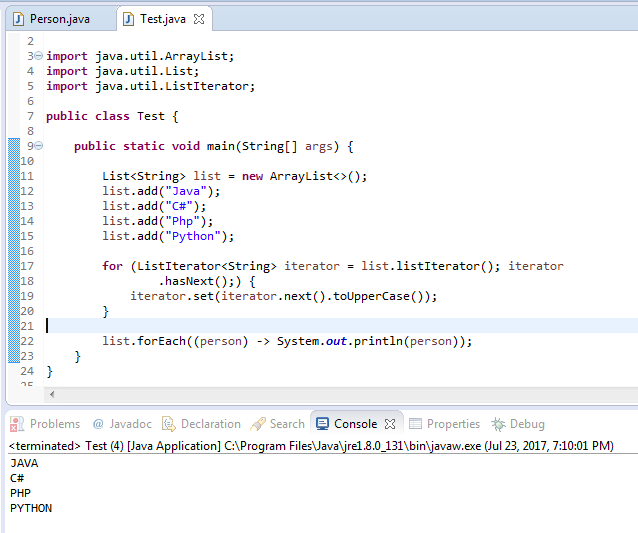
After the introduction of lambda in java, collections APIs are improved and new API methods are exposed.

**New APIs using Functional Interface**

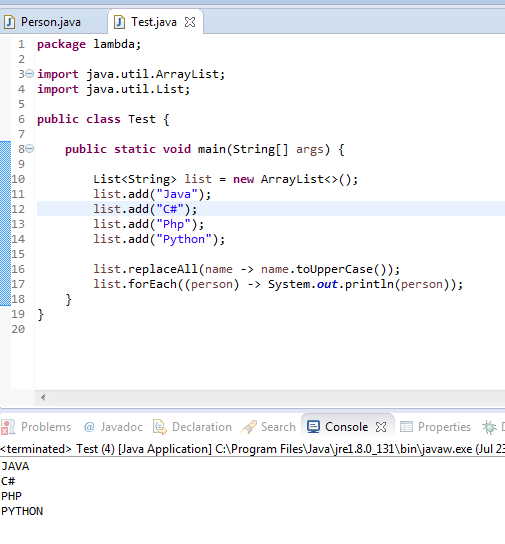
****

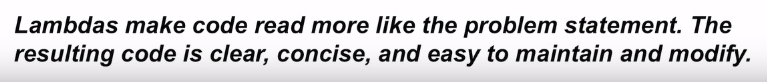
**Convert all the words of a list in uppercase**

**Before lambda**



**After lambda**



****

**Benefits**

It enables the functional programming.

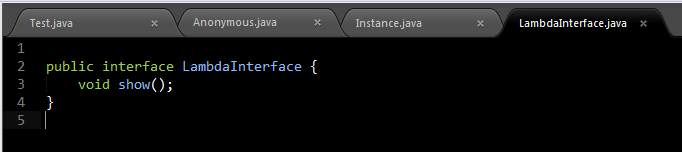
It eliminates boiler plate code.

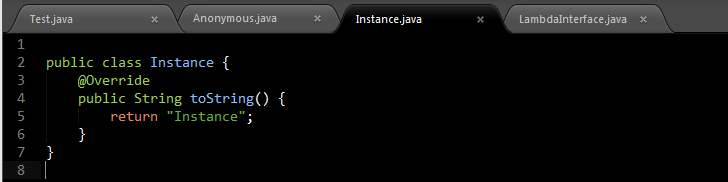
It supports for parallel processing.

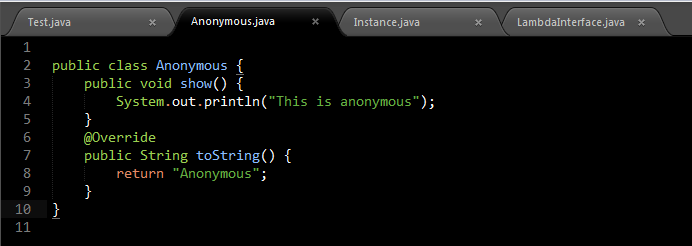
It does not create class files for anonymous class being implemented.

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

|  |  |
| --- | --- |
| **Anonymous Inner class** | **Lambda expression** |
| It is a class without a name | It is a function without a name |
| Anonymous inner class can extend abstract and concrete classes | λ – expression cannot extend abstract and concrete classes |
| Anonymous inner class can implement an interface that contains any number of methods | λ – expression can implement an interface which contains single abstract method |
| Inside anonymous inner class, we can declare instance variables | Inside λ – expression we cannot declare instance variables whatever variables are declared are called local variable |
| anonymous inner class can be instantiated | λ – expression cannot be instantiated |
| Inside anonymous inner class, this always refers current anonymous inner class object but not outer class object | Inside λ – expression, this always refers current outer class object |
| Anonymous inner class is best choice if we want to handle multiple methods | λ – expression is the best choice if we want to handle interface with single abstract method(Functional interface) |
| For anonymous inner class at compilation time a separate .class file will be generated | For λ – expression at compilation time no separate .class file will be generated |
| Memory will be allocated on demand, whenever we are creating object | λ – expression will reside in permanent memory of JVM(method area) |

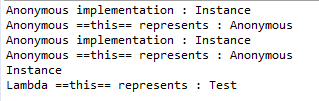




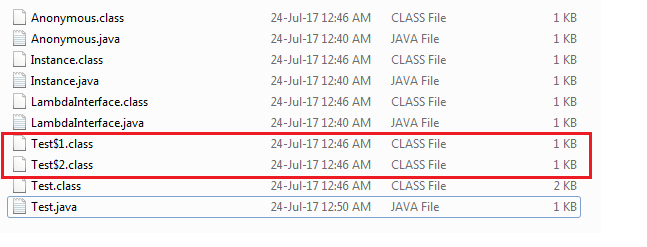




**Output**



**Generated class files**



**How is a lambda expression represented at runtime?**

By a lambda object; both the lambda object and its type are dynamically created by the virtual machine at runtime.

**What is the target type of a lambda expression?**

A type to which the lambda expression can be converted in a given context; the target type must be a functional interface type.

**What is a functional interface?**

An interface with a single abstract method.

**What do we need lambda expressions in Java for?**

To enable convenient use of the overhauled collection framework in general and it parallel bulk operations in particular.

**What is a bulk operation?**

An operation that concern many or all elements in a sequence.