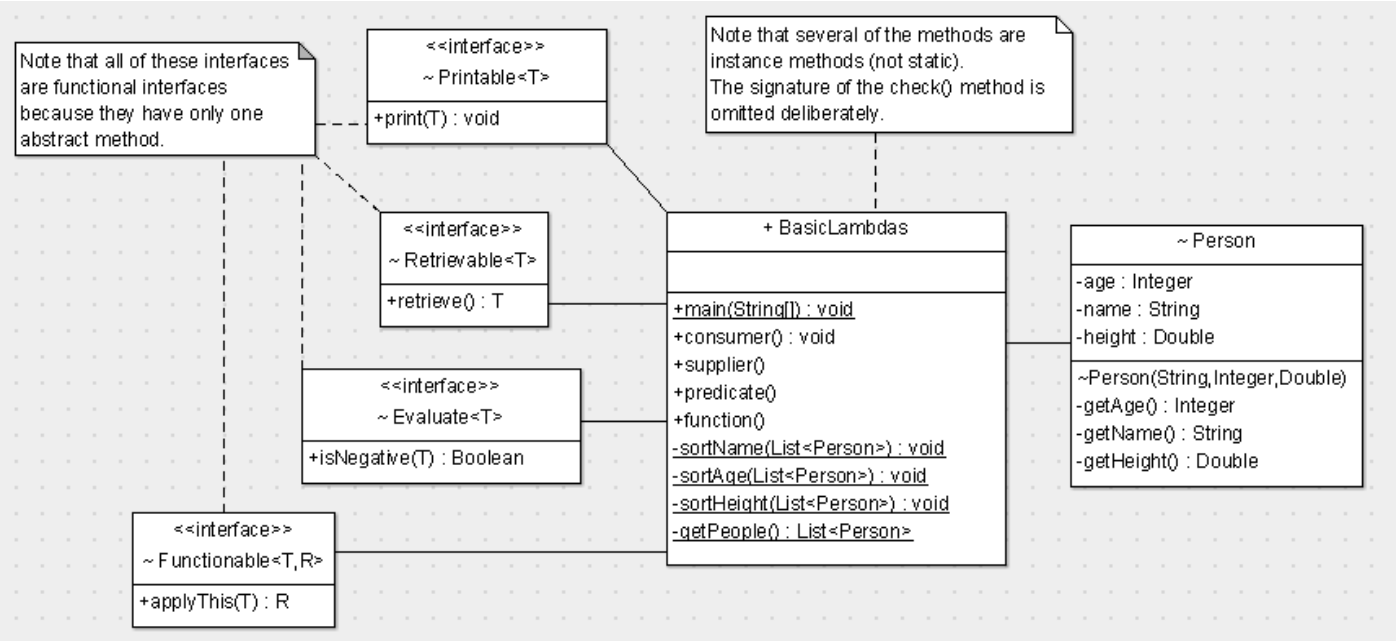
Lambdas Lab



PART 1

1. In main() invoke the **consumer()** method; in consumer() do the following:
    a) Using a lambda expression, implement the *Printable* interface (typed for *String*). The relevant method just prints out the String argument it receives. Invoke the relevant method, passing in "Printable lambda".
    b) Using both a lambda expression and a method reference, implement 1a using a *Consumer*.
2. In main() invoke the **supplier()** method; in supplier() do the following:
    a) Using a lambda expression, implement the *Retrievable* interface (typed for *Integer*). The relevant method just returns 77. Invoke the relevant method.
    b) Using a lambda expression, implement 2a using a *Supplier*.
3. In main() invoke the **predicate()** method; in predicate() do the following:
    a) Using a lambda expression, implement the *Evaluate* interface (typed for *Integer*). The relevant method returns *true* if the argument passed is < 0, otherwise it returns *false*. Invoke the relevant method twice – the first time pass in -1 and the second time pass in +1
    b) Using a lambda expression, implement 3a using a *Predicate*.
    c) Declare a generically-typed check() method (not in UML). The first parameter is generic and the second parameter is a *Predicate*, also generically typed. The check() method returns true/false. Invoke the check() method with the following *Predicate* lambda expressions:
        - we want to know if a number is even (true) – invoke check() with 4 and 7 (true and false).
        - we want to know if a String begins with "Mr." – invoke check() with "Mr. Joe Bloggs" and "Ms. Ann Bloggs"
        - we want to know if a person is an adult (age >= 18) – invoke check() with "Mike" who is 33 and 1.8 (metres assumed) in height; and "Ann" who is 13 and 1.4 (metres) in height.
4. In main() invoke the **function()** method; in function() do the following:
    a) Using a lambda expression, implement the *Functionable* interface - the input type is *Integer* and the return type is *String*. The relevant method returns the number passed in appended to the *String* "Number is: ". Invoke the relevant method passing in 25.
    b) Using a lambda expression, implement 4a using a *Function*.

PART 2

Given the following implementation of the getPeople() method:

```
private static List<Person> getPeople() {
        List<Person> result = new ArrayList<>();
        result.add(new Person("Mike", 33, 1.8));
        result.add(new Person("Mary", 25, 1.4));
        result.add(new Person("Alan", 34, 1.7));
        result.add(new Person("Zoe", 30, 1.5));
        return result;
    }
```

5. In main(), invoke the **getPeople()** – store the result in a variable named listPeople.
6. In main(), invoke the **sortAge()** method passing down listPeople; in sortAge() do the following:
    a) Using the Iterable sort() method (note: *List* extends *Iterable*), and the Comparator.comparing() method, sort the Person objects in ascending age order. Note that the argument to Comparator.comparing() requires a Function (In, Out) that returns a Comparable (a class that implements Comparable). From that, the comparing() method generates a Comparator that it passes to the sort() method.
        - Note that as of Java 8, the List interface supports the sort() method directly so there is no need to use the Collections.sort(): i.e. instead of *Collections.sort(list, comparatorRef);* we now have *list.sort(comparatorRef);*
    b) Output the sorted list using the Iterable forEach() method passing in a lambda expression.
7. In main(), invoke the **sortName()** method passing down listPeople; in sortName() do the following:
    a) As in 6a except sort the Person objects in ascending name order.
    b) Output the sorted list using the Iterable forEach() method passing in a lambda expression.
8. In main(), invoke the **sortHeight()** method passing down listPeople; in sortHeight() do the following:
    a) As in 6a except sort the Person objects in ascending height order.
    b) Output the sorted list using the Iterable forEach() method passing in a lambda expression.
9. Refactor 6b, 7b and 8b to use method references instead of lambda expressions.