JAVA 8 Lambda

BY:

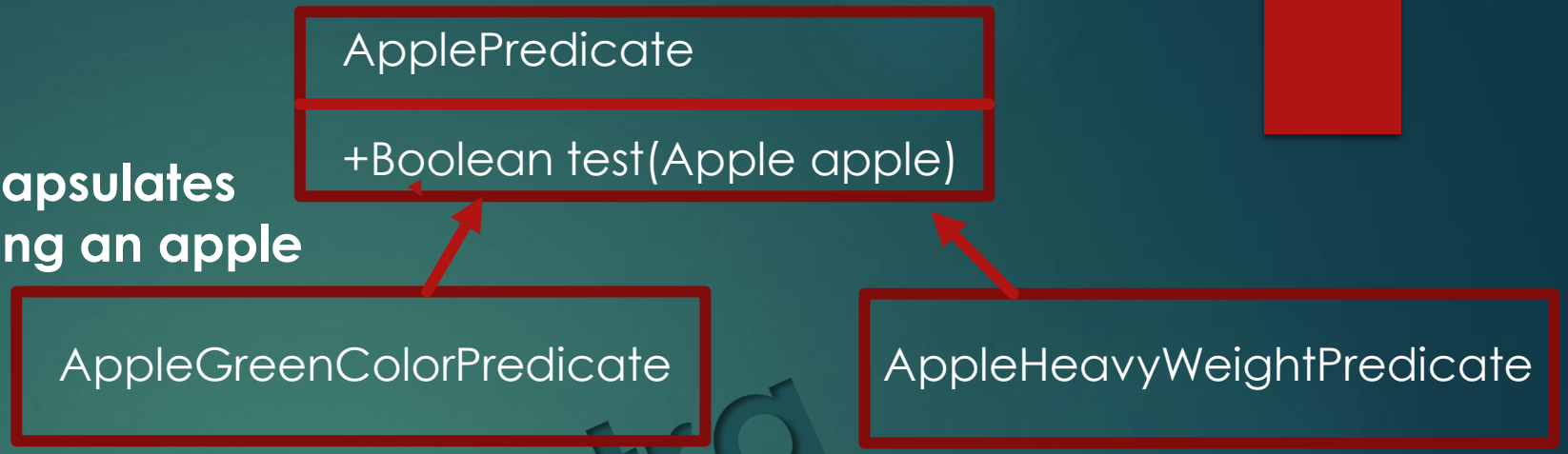
SUJATA BATRA

Behaviour Parameterization

- Behavior parameterization is the ability for a method to take multiple different behaviors as parameters and use them internally to accomplish different behaviors.
- Behavior parameterization lets us make your code more adaptive to changing requirements and saves on engineering efforts in the future.

Example

ApplePredicate encapsulates a strategy for selecting an apple




```
//Behaviour Parameterization
public static List<Apple> filterApple(List<Apple> inventory,ApplePredicate p){
        List<Apple> result =new ArrayList<>();
        for(Apple apple:inventory){
                if(p.test(apple)){
                    result.add(apple);
                }
         }
          return result;
}
```

Note: The only code really matters is implementation of the test method. Unfortunately,because filterApple method can only take objects,we have to wrap code inside an ApplePredicate object.

Anonymous classes

- Declare and instantiate a class at the same time.
- Don't have name.
- Used to reduce verbosity and time.

Note: Passing code is a way to give new behaviors as arguments to a method. But it's verbose prior to Java 8.
Anonymous classes helped a bit before Java 8 to get rid of the verbosity associated with declaring multiple concrete classes for an interface that are needed only once.

- The Java API contains many methods that can be parameterized with different behaviors, which include sorting, threads etc.

Lambdas

Sujata Batra

Lambda Introduction

- Representation of an anonymous function : it doesn't have a name, but it has a list of parameters, a body, a return type, and also possibly a list of exceptions that can be thrown.
- Can be passed around as a parameter thus achieving **behavior parameterization.**
- Let you pass code in a concise way.

- An instance of a lambda can be assigned to any **functional interface** whose single abstract method's definition matches the definition of the lambda.
- Lamda can be
  - assigned to variables
  - passed to functions

What are Lambda's good for

- Forms the basis of functional programming
- Make parallel programming easier
- Write more compact code
- Richer data structure collection
- Develop cleaner APIs

Syntax

- Lambda expression is composed of parameters,an arrow and a body.
- parameter -> expression body
  Or
  (parameters) -> { statements; }
  or
  () -> expression

- Following are Some examples of Lambda


```
(int a, int b) ->    a * b                    // takes two integers and returns their multiplication
(a, b)          ->   a - b                    // takes two numbers and returns their difference
() -> 99                                      // takes no values and returns 99
(String a) -> System.out.println(a)           // takes a string, prints its value to the console, and returns nothing
a -> 2 * a                                    // takes a number and returns the result of doubling it
c -> { //some complex statements }            // takes a collection and do some procesing
```

Rules for writing lambda expressions

- A lambda expression can have zero, one or more parameters.
- The type of the parameters can be explicitly declared or it can be inferred from the context.
- Multiple parameters are enclosed in mandatory parentheses and separated by commas. Empty parentheses are used to represent an empty set of parameters.
- When there is a single parameter, if its type is inferred, it is not mandatory to use parentheses. e.g. a -> return a*a.
- The body of the lambda expressions can contain zero, one or more statements.
- If body of lambda expression has single statement curly brackets are not mandatory and the return type of the anonymous function is the same as that of the body expression. When there is more than one statement in body than these must be enclosed in curly brackets.

Functional Interface

Functional Interfaces in Java 8

- **A functional interface**, is an interface which has only a single abstract method.

- **@FunctionalInterface annotation**
  - used to explicitely specify that a given interface is to be treated as a functional interface.
  - is an *informative annotation.*

Custom Or User defined Functional Interfaces

- Interfaces defined by the user and have a single abstract method. These may/may not be annotated by @FunctionalInterface.

```
package com.sujata.java8training;

@FunctionalInterface
public interface MyCustomFunctionalInterface {

    //This is the only abstract method.Hence, this
    //interface qualifies as a Functional Interface
    public void firstMethod();

}
```

Pre-existing functional interfaces in Java prior to Java 8

- interface java.lang.Runnable{
  void run();
  }
- interface java.util.Comparator<T>{
  int compare(T o1,T o2)
  }