**COSC60220220128Functional**

ConsumerImpl.java

public class ConsumerImpl<T> implements Consumer<T> {  
  
 @Override  
 public void accept(T t) {  
 System.*out*.println(t);  
 }  
}

ListedDemo.java

public class ListedDemo {

private static List<String>*list*= Arrays.*asList*("One", "two", "Three", "four" );  
  
public static void main(String[] args) {

//pass unnamed instance to forEach  
*list*.forEach(new ConsumerImpl<>());  
  
System.*out*.println("+++++ create a class on the fly ++++++");  
  
//create a class on the fly  
*list*.forEach(new Consumer<String>() {  
 @Override  
 public void accept(String s) {  
 System.*out*.println(s);  
 }  
});  
  
System.*out*.println("++++++ lambda notation +++++++++++++");  
  
//with lambda notation  
// NO-public NO-void NO\_accept(NO-T t) {  
 // System.out.println(s); } t-> sout(t);  
*list*.forEach(t -> System.*out*.println(t));  
  
System.*out*.println("+++++ lambda notation with variable ++++++++");  
  
Consumer consumer **= t -> System.*out*.println(t)**;  
*list*.forEach(consumer);  
  
  
System.*out*.println("+++++ Hash Set and make Lambda expression ++++++++");  
  
HashSet<Integer> myHash= new HashSet<>();  
myHash.add(1);  
myHash.add(2);  
myHash.add(4);  
myHash.add(7);  
myHash.add(3);  
myHash.add(5);  
  
System.*out*.println(myHash.toString());  
  
// public boolean test(T t){ return (t>1);) => Lambda = t->t>1  
myHash.removeIf(t->t>3);  
  
System.*out*.println(myHash.toString());  
}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

FunctionInput.java

public class FunctionInput {  
 private static List<String> *list*= Arrays.*asList*("One","two","Three","four");  
  
 public static void main(String[] args) {  
  
 //This is a static variable but second approach is parametrize it and behaviour with any input is correct  
 for (String element:*list*)  
 System.*out*.println(element.toUpperCase());  
  
 *list*.forEach(s-> System.*out*.println(*processString*(t->t.toUpperCase(),s)));  
  
 }  
 //method with different passing  
 // first input is function , second is var  
 public static String processString(Function<String,String> operation, String target){  
 return operation.apply(target);  
 }  
}