**Revise for reduce():**

List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5, 6);

int result = numbers

.stream()

.reduce(0, (subtotal, element) -> subtotal + element);

assertThat(result).isEqualTo(21);

1. If these are the lists in the stream: [“Hello”, “there”], [“goodbye”, “again”], then the output of your combine method would be [“Hello”, “there”, “goodbye”, “again”].  
     
   Your solution *must make use of the reduce method for Streams.*

**public class** Problem4 {  
 *//IMPLEMENT - you must use the reduce operation on Streams* **public static** <T> ArrayList<T> combine(Stream<ArrayList<T>> stream){

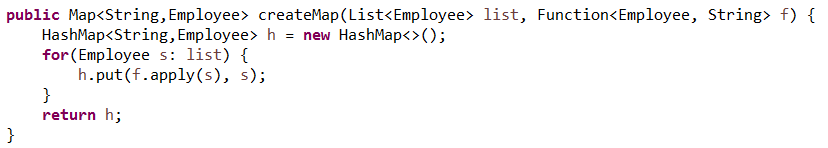
**return new** ArrayList<T>();   
 }  
   
 **public static void** testCombine() {  
 ArrayList<ArrayList<String>> mainList = **new** ArrayList<>();  
 ArrayList<String> list1 = **new** ArrayList<String>() {  
 {  
 add(**"hello"**);add(**"there"**);  
 }  
 };  
 ArrayList<String> list2 = **new** ArrayList<String>() {  
 {  
 add(**"goodbye"**);add(**"there"**);  
 }  
 };  
 mainList.add(list1);  
 mainList.add(list2);  
 System.***out***.println(*combine*(mainList.stream()));  
 }  
 **public static void** main(String[] args) {  
 *testCombine*();  
 }  
}

1. Implement a method

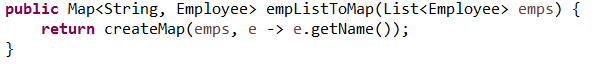
**public** Set<String> union(List<Set<String>> sets)

by creating a stream pipeline that transforms a list of sets (of type String) into the union of those sets. Make use of the reduce method for streams.   
  
*Example*: The union method should transform the list [{“A”, “B”}, {“D”}, {“1”, “3”, “5”}] to the set {“A”, “B”, “D”, “1”, “3”, “5”}.

1. In the code below, an implementation of a method createMap is shown. This method allows you to create, from a given list of Employees, a HashMap (returned as the interface type Map) in which the keys are names of Employees and the values are Employee objects.



The createMap method is applied in the following method:



For instance, starting from a list of Employees, we can generate a map matching Employee names to Employee objects:

List<Employee> list = Arrays.asList(new Employee(101, "Joe", 50000),new Employee(102, "Jim", 60000));

Map<String, Employee> map = empListToMap(list);

Now:

map.get("Joe") returns the Employee object having name "Joe"

map.get("Jim") returns the Employee object having name "Jim"

For this problem the objective is to generalize the createMap method as much as possible. You will want to generalize from Employee and String in the createMap method shown above to more general types.

Test how successful you are by seeing if your final createMap method can be used to implement each of the following two methods (see the helperclasses package to see the definitions of Account, CheckingAccount and Person).

**public** Map<Object, Employee> empListToMap(List<Employee> emps) {

**return** createMap(emps, (Person p) -> p.getName());

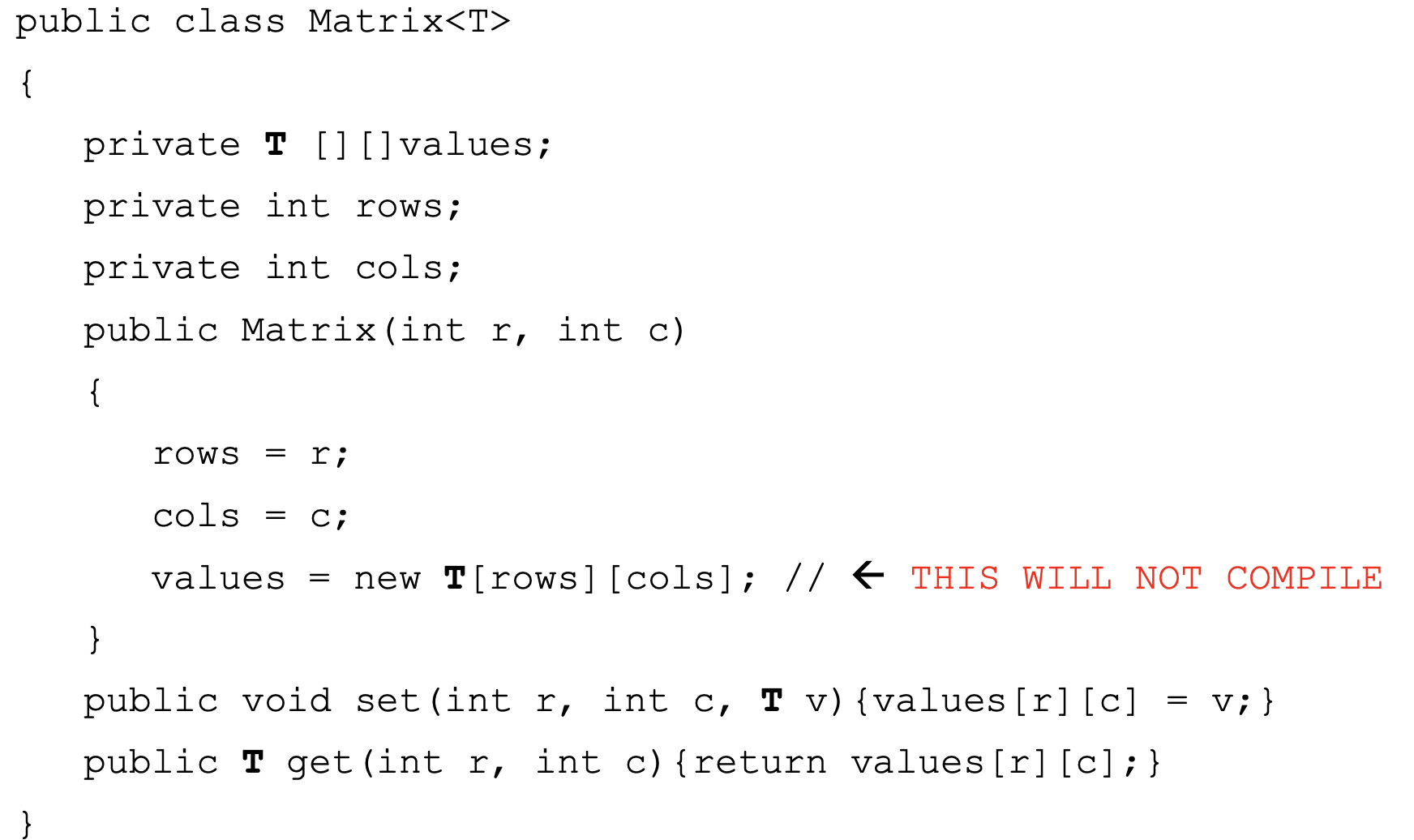
}

**public** Map<Number, CheckingAccount> checkingAcctListToMap(List<CheckingAccount> list) {

**return** createMap(list, (Account a) -> a.getAcctId());

}

1. Solve the given problem using generics.



1. The given code does not give the output in the following style.

**Source Code**:

**class** Callme {  
 **void** call(String msg) {  
 System.***out***.print(**"["** + msg);  
 **try** {  
 Thread.*sleep*(1000);  
 } **catch**(InterruptedException e) {  
 System.***out***.println(**"Interrupted"**);  
 }  
 System.***out***.println(**"]"**);  
 }  
}  
**class** Caller **implements** Runnable {  
 String **msg**;  
 Callme **target**;  
 Thread **t**;  
 **public** Caller(Callme targ, String s) {  
 **target** = targ;  
 **msg** = s;  
 **t** = **new** Thread(**this**);  
 **t**.start();  
 }  
 **public void** run() {  
 **target**.call(**msg**);  
 }  
}  
**public class** Synch {  
 **public static void** main(String args[]) {  
 Callme target = **new** Callme();  
 Caller ob1 = **new** Caller(target, **"Hello"**);  
 Caller ob2 = **new** Caller(target, **"Synchronized"**);  
 Caller ob3 = **new** Caller(target, **"World"**);  
 *// wait for threads to end* **try** {  
 ob1.**t**.join();  
 ob2.**t**.join();  
 ob3.**t**.join();  
 } **catch**(InterruptedException e) {  
 System.***out***.println(**"Interrupted"**);  
 } }  
}

Apply synchronization and produce the output in the following pattern.

[Hello]

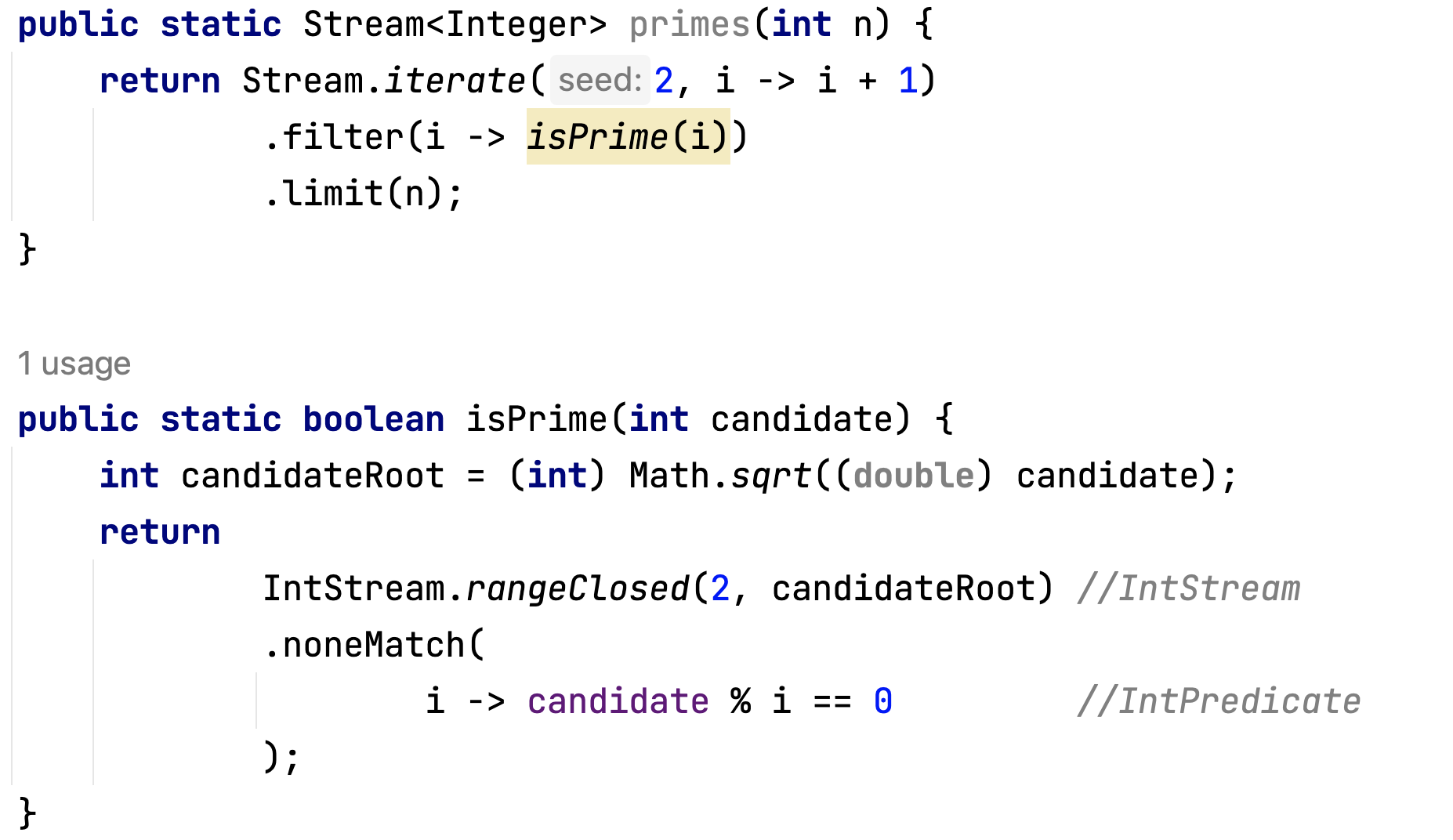
[World]

[Synchronized]

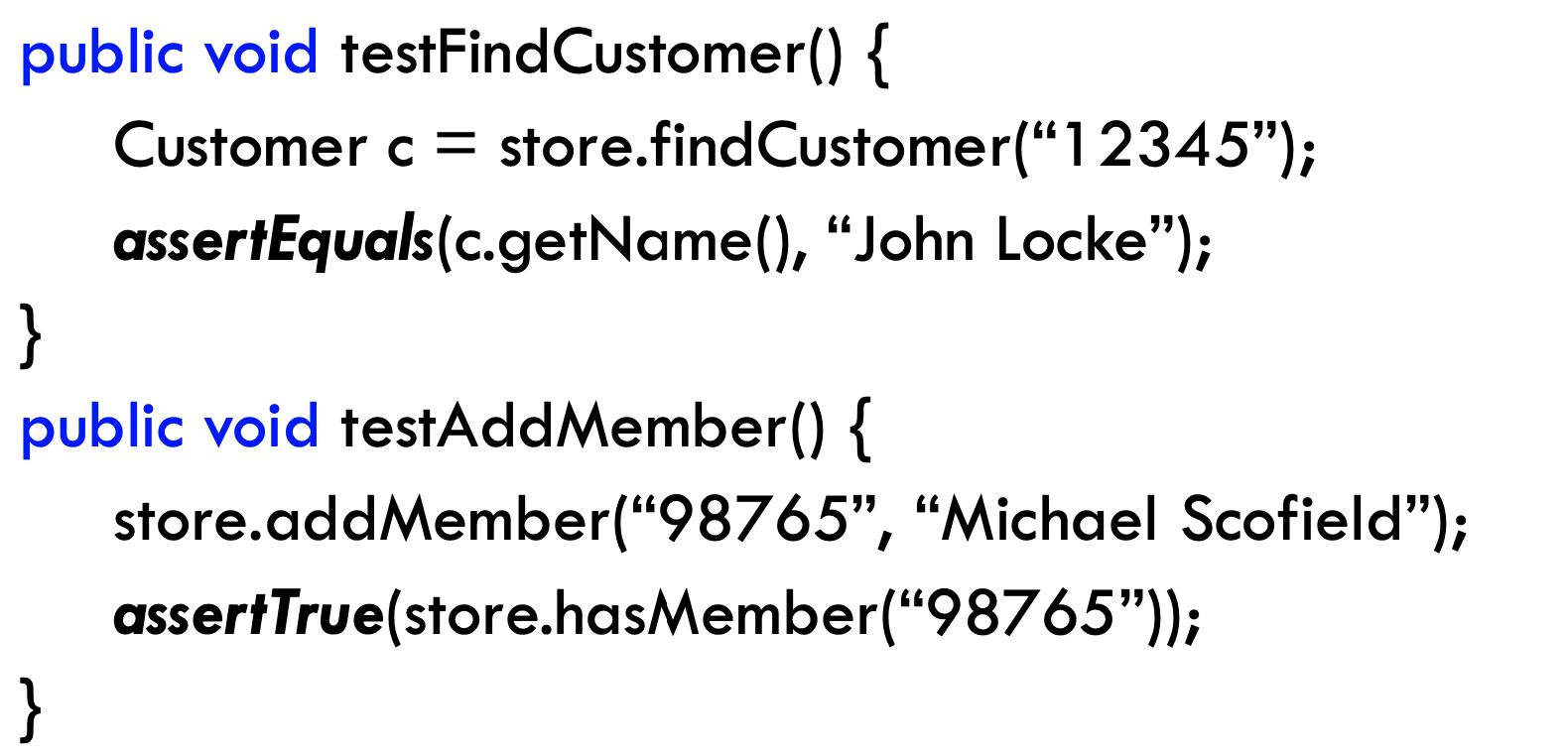
1. Create a user defined annotation and annotate the class. Get the annotation information from a main class.
2. Change the given imperative coding to functional programing.

**public double** calculateAverage(List<Employee> employeeList) {  
 **int** sum = 0;  
 **int** count = 0;  
 **for** (**int** i = 0; i < employeeList.size(); i++) {  
 Employee employee = employeeList.get(i);  
 sum += employee.getSalary();  
 count++;  
 }  
 **return** (**double**) sum / count;  
}

1. Compute the list of all primes.



1. Review assertTrue(), assertEquals() and assert().



1. Review lab assignments – interface in Java 8
2. Overriding equals method using instanceOf strategy and class strategy.
3. Overriding hashCode