Work on this assignment by yourself.

Write the code for the assignment below and upload the zipped src folder to the learning hub (learn.bcit.ca 🡪 Activities 🡪 Assignments 🡪 Assignment 1) before the start of lesson 7.

Include your full name at the top of each file, using a Java Javadoc   
/\*\* @author \*/ tag (for example: /\*\* @author Jason Wilder \*/).

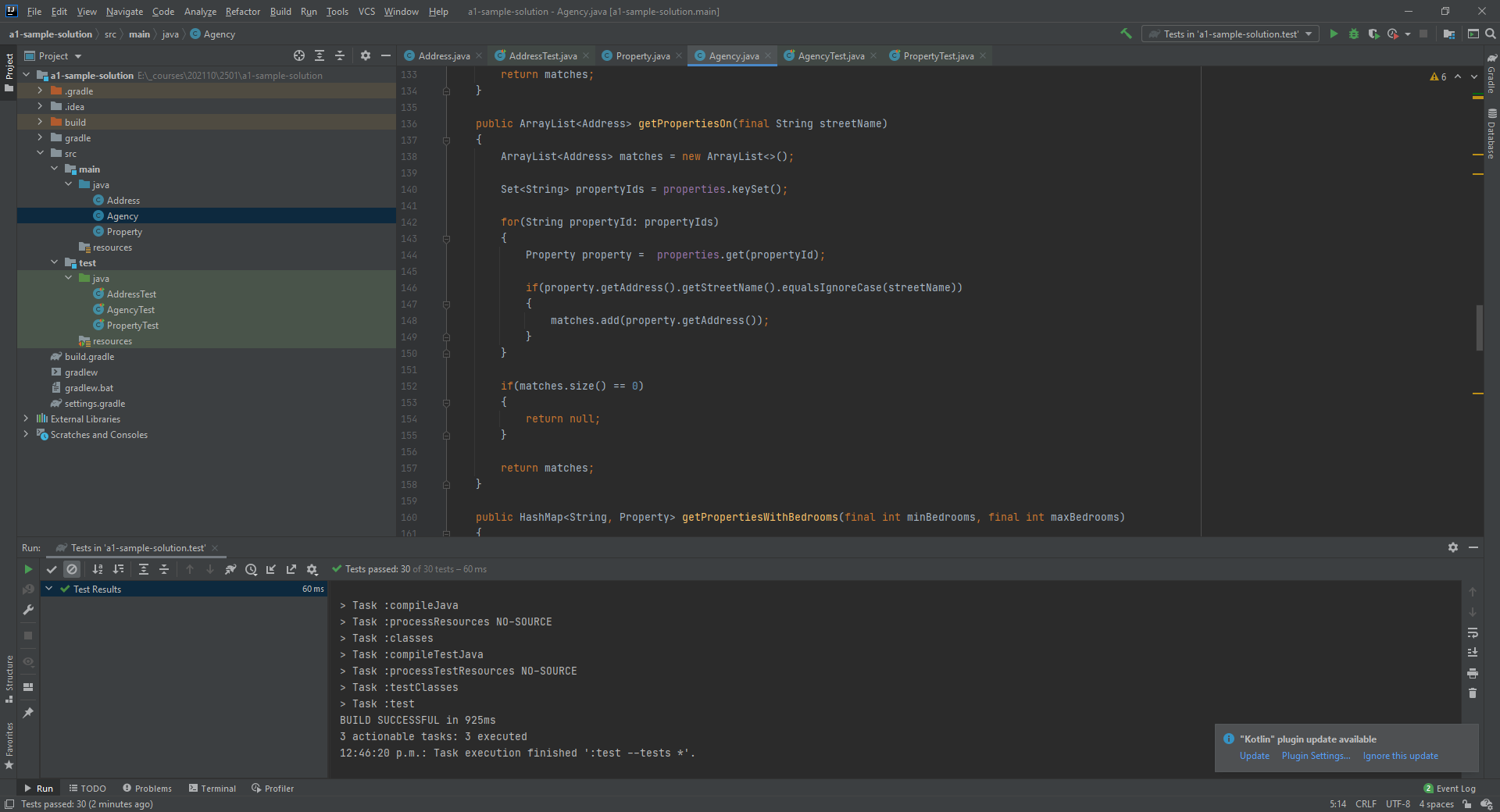
Create the following classes along with the specified instance data and methods. Make sure you use   
/\*\* Javadoc comments \*/ to comment all public classes, constructors, and methods (with the @author, @version, @return, and @param tags as appropriate). Make sure the constructor validates data as specified. All constructors and mutators must throw an IllegalArgumentException when parameters do not match the expected format (listed below). Make all data (including parameters) final if it is not expected/desired to change. Avoid magic numbers; e.g. use public static final double PI = 3.14 instead of simply using 3.14). To have the following test and main folders created automatically for you, use Gradle. To use gradle, all one needs to do is to select gradle when starting a new project. The test and main folders are created automatically.

| **Class** | **Instance Variables** | **Methods** |
| --- | --- | --- |
| Address | **Unit number** (String: must be one to four characters)  **Street number** (int must be 0 to 999999)  **Street name** (String: must be one to twenty characters)  **Postal code** (String: must be either length 5 or length 6)  **City** (String: must be one to thirty characters) | *Get* methods for all instance variables |
| Property | **Price in USD** (double, must be positive)  **Address** (Address, not null)  **Number of bedrooms** (int: must be 1 to 20)  **Swimming pool** (boolean)  **Type** (String: must be one of: “residence”, “commercial”, or “retail”)  **Property ID** (String: must be one to six characters) | *Get* methods for all instance variables  *Set* method for price |
| Agency | **Name** (String, 1 to 30 characters)  **Properties** (HashMap of properties; key is property id, value is a Property) | **addProperty(property):** adds the (non-null) property to the HashMap  **removePropery(propertyId):** removes the property whose ID matches the parameter, from the HashMap  **getProperty(propertyId):** returns the property whose ID matches the parameter, from the HashMap (or null if there is no match)  **getTotalPropertyValues()**: returns the total amount in USD of all Properties  **getPropertiesWithPools()**: returns an ArrayList of such Properties…or null if there are none  **getPropertiesBetween(minUsd, maxUsd)**: returns an array of properties whose price falls in the range specified by the parameters…or null if there are none  **getPropertiesOn(streetName)**: returns an ArrayList of addresses which are on the specified street…or null if there are none  **getPropertiesWithBedrooms(minBedrooms, maxBedrooms)**: returns a HashMap of properties (key is property id, value is the Property) whose number of bedrooms falls in the range specified by the parameters…or null if there are none. Note that the *order* of the properties may differ from since a HashMap doesn’t store “in order,” but the contents must otherwise match those in the sample run in the tests.  **getPropertiesOfType(propertyType)**: returns an ArrayList of Strings, with all of the information about every property (one property per line) that is of the specified type (e.g. “commErciAl”: be case insensitive) in the exact format of:  Type: COMMERCIAL  1) Property 9999: unit #9 at 99 Gretzky Way T6V7H3 in Toronto (1 bedroom): $99999.  2) Property 678T: 1515 Main Street V8Y7R3 in West Vancouver (2 bedrooms plus pool): $4000000.  3) Property A1212: unit #7h at 1500 Railway Avenue V9V5V4 in Richmond (4 bedrooms): $840000.  Note that the sample output above is exactly what should be returned for the data shown below; it must create similar sentence structures for **any** property of **any** type. Notice the capitalization of various parts of the string (see above) versus how it was stored (see below).  If there are NO properties of the specified type the output must be as follows:  Type: RETAIL  <none found> |
| Sample run (e.g. in the unit tests). |  | The unit tests create an agency named “BCIT Ltd” with the following properties:   |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | | **unit** | **st #** | **st name** | **city** | **postal** | **# bed** | **type** | **id** | **$** | | 1a | 777 | 56th avenue | surrey | v7n2m8 | 2 | res | abc123 | 499000 | |  | 123 | main street | west vancouver | v7r2g2 | 5 | res  pool | xyz789 | 5999999 | |  | 456 | elm street | los angeles | 90210 | 6 | res  pool | 777def | 2500000 | | 44 | 1111 | maple street | vancouver | v8y3r5 | 1 | ret | 876tru | 1000000 | | 9 | 99 | gretzky way | toronto | t6v7h3 | 1 | com | 9999 | 99999 | | b | 711 | country road | maple ridge | v8h5f5 | 3 | res | mr6789 | 740100 | |  | 8785 | pinnacle avenue | north vancouver | v9u3h3 | 20 | res  pool | 78444a | 15000000 | |  | 800 | elm street | los angeles | 90557 | 10 | res | mmm33 | 7100000 | |  | 1515 | main street | west vancouver | V8Y7R3 | 2 | com  pool | 678T | 4000000 | | 6 | 60 | 60th street | burnaby | v8u9b1 | 2 | ret  pool | y6yyy | 700000 | | 7h | 1500 | railway avenue | richmond | v9v5v4 | 4 | com | A1212 | 840000 | |  | 333 | elm street | los angeles | 90111 | 3 | res | 9000a | 1600000 | |

# Testing and Grading your work

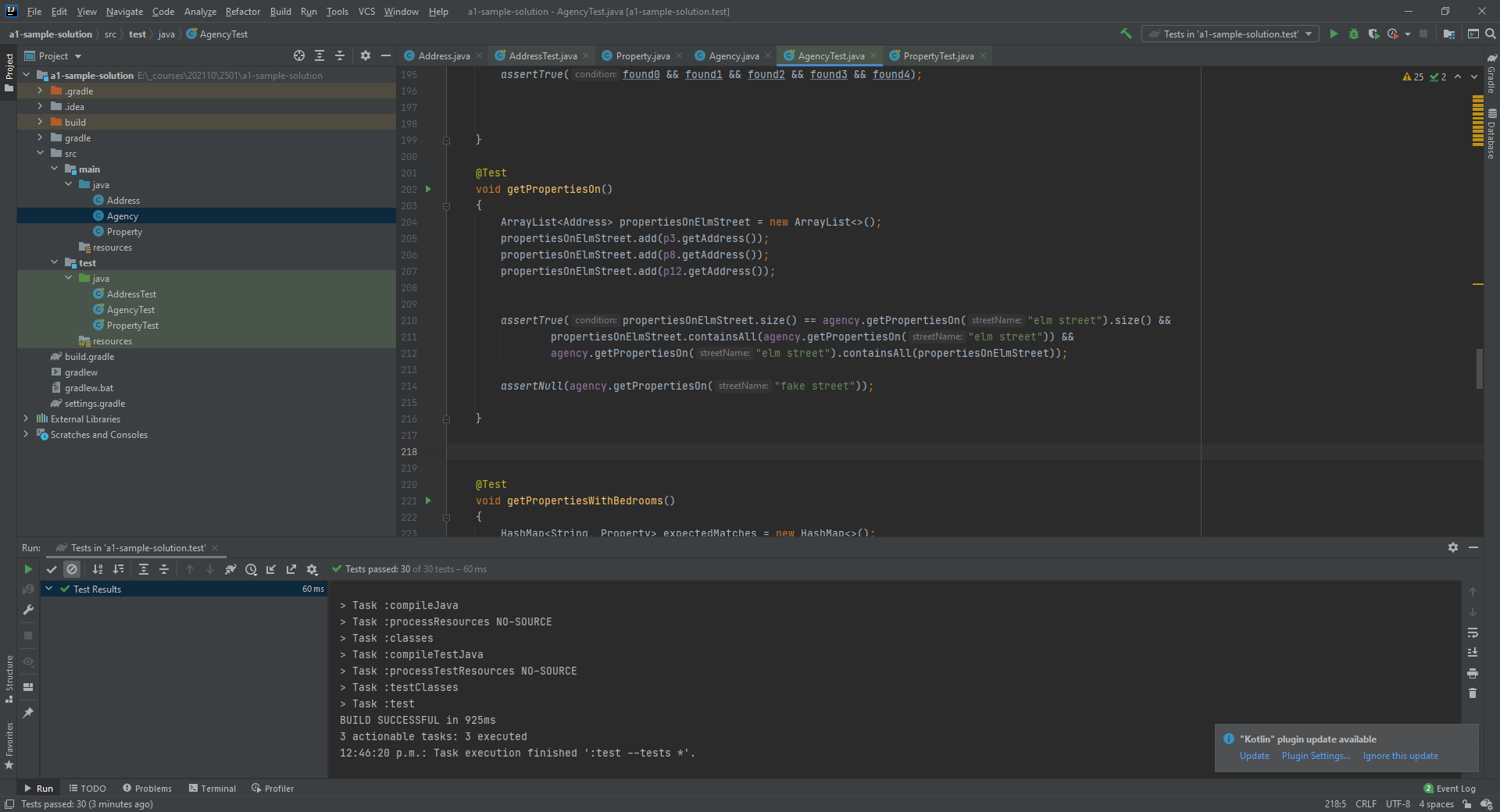
Below are tests for each of your three classes.

Copy the code into three files and put them into a folder named “java” inside another folder named “test” as in the screen shot here:



By the way…one of the Agency methods is displayed there for you too. It can act as a hint for other methods.

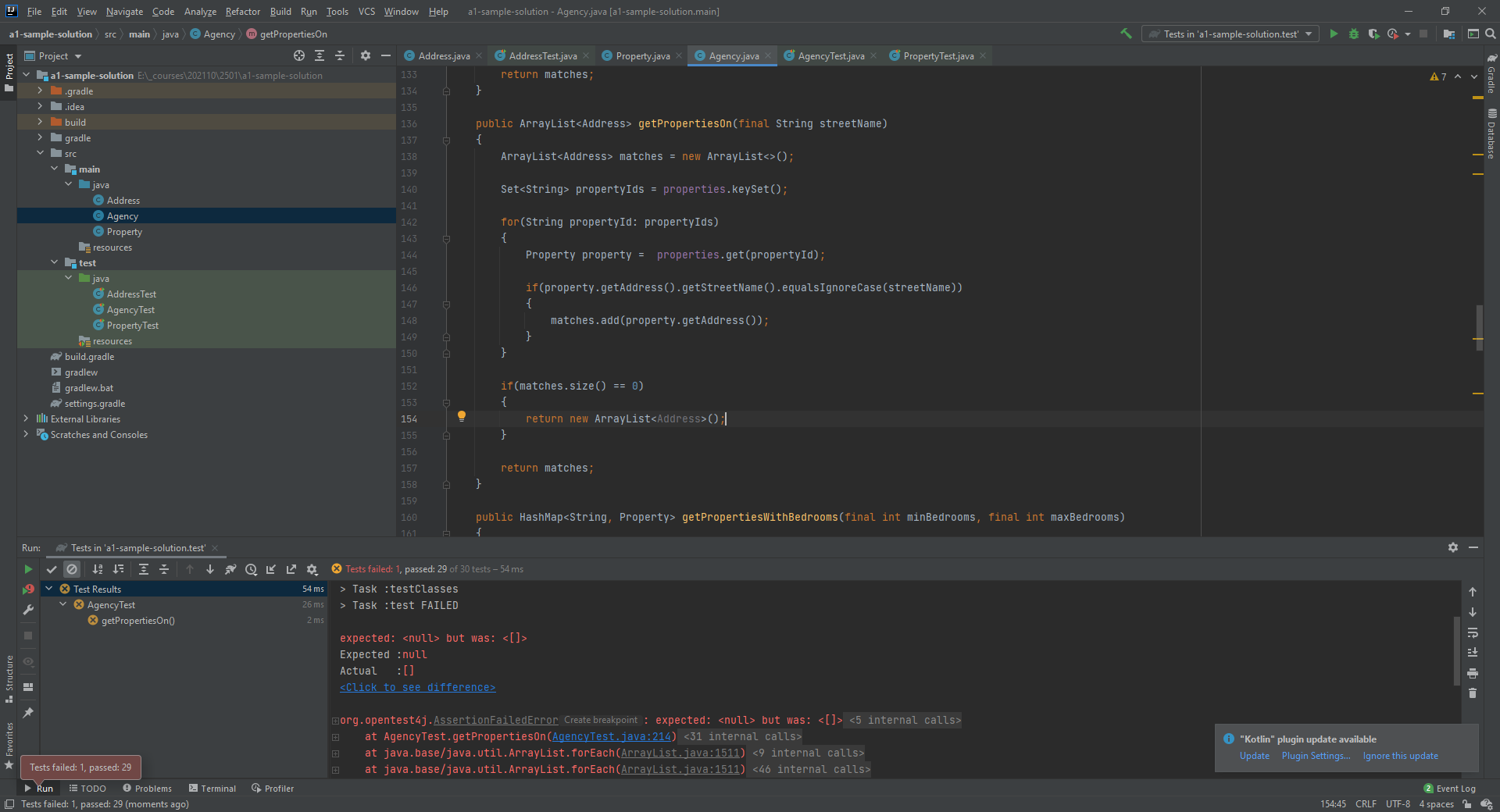
At the top-right of the IDE there is a dropdown menu saying “Tests in ‘a1-sample-solution-test’” (which is the name of my project). Pressing the green arrow will run all 30 of the tests for the Address, Property, and Agency classes. So far, if you copy the code provided for the “getPropertiesOn()” method, you should score 1/30 because it passes the AgencyTest.java class’s “getPropertiesOn()” test, here:



At the bottom-left of the IDE in this screenshot it says “Tests passed: 30” which means I scored 30/30 for assignment 1. In addition to passing these 30 unit tests, your instructor will give grades for:

* Good naming
* Making sure instance variables are private
* No magic numbers
* Good indentation and style.

If I had made a mistake (see line 154 below) then I would *not* have passed all 30 tests:



And jUnit would tell me what was wrong and where (line 214 of AgencyTest.java failed).

Pass all 30 tests. For your easy reading and referral, the unit test code is provided below (but note that the files themselves are also on the learning hub for your to download and use):

import org.junit.jupiter.api.AfterEach;  
import org.junit.jupiter.api.BeforeEach;  
import org.junit.jupiter.api.Test;  
  
import static org.junit.jupiter.api.Assertions.\*;  
  
class AddressTest  
{  
 private Address address1;  
 private Address address2;  
  
 @BeforeEach  
 void setUp()  
 {  
 address1 = new Address("1a", 777, "56th avenue", "v7n2m8", "surrey");  
 address2 = new Address(null, 123, "main street", "v7r2g2", "west vancouver");  
 }  
  
 @AfterEach  
 void tearDown()  
 {  
 address1 = null;  
 address2 = null;  
 }  
  
 @Test  
 void getUnitNumber()  
 {  
 *assertEquals*("1a", address1.getUnitNumber());  
 *assertNull*(address2.getUnitNumber());  
 }  
  
 @Test  
 void getStreetNumber()  
 {  
 *assertEquals*(777, address1.getStreetNumber());  
 *assertEquals*(123, address2.getStreetNumber());  
 }  
  
 @Test  
 void getStreetName()  
 {  
 *assertEquals*("56th avenue", address1.getStreetName());  
 *assertEquals*("main street", address2.getStreetName());  
 }  
  
 @Test  
 void getPostalCode()  
 {  
 *assertEquals*("v7n2m8", address1.getPostalCode());  
 *assertEquals*("v7r2g2", address2.getPostalCode());  
 }  
  
 @Test  
 void getCity()  
 {  
 *assertEquals*("surrey", address1.getCity());  
 *assertEquals*("west vancouver", address2.getCity());  
 }  
  
 @Test  
 void getExpectedExceptionsUnitNumber()  
 {  
 IllegalArgumentException ex = *assertThrows*(IllegalArgumentException.class, ()->{  
 address1 = new Address("",777, "56th avenue", "v7n2m8", "surrey");  
 });  
 *assertTrue*(ex.getMessage().equals("Invalid unit number: "));  
  
 ex = *assertThrows*(IllegalArgumentException.class, ()->{  
 address1 = new Address("11111",777, "56th avenue", "v7n2m8", "surrey");  
 });  
 *assertTrue*(ex.getMessage().equals("Invalid unit number: 11111"));  
 }  
  
 @Test  
 void getExpectedExceptionsStreetNumber()  
 {  
 IllegalArgumentException ex = *assertThrows*(IllegalArgumentException.class, ()->{  
 address1 = new Address("1a",-1, "56th avenue", "v7n2m8", "surrey");  
 });  
 *assertTrue*(ex.getMessage().equals("Invalid street number: -1"));  
  
 ex = *assertThrows*(IllegalArgumentException.class, ()->{  
 address1 = new Address("1a",1000000, "56th avenue", "v7n2m8", "surrey");  
 });  
 *assertTrue*(ex.getMessage().equals("Invalid street number: 1000000"));  
 }  
  
 @Test  
 void getExpectedExceptionsStreetName()  
 {  
 NullPointerException ex2 = *assertThrows*(NullPointerException.class, ()->{  
 address1 = new Address("1a",777, null, "v7n2m8", "surrey");  
 });  
 *assertTrue*(ex2.getMessage().equals("Invalid street name: null"));  
  
 IllegalArgumentException ex = *assertThrows*(IllegalArgumentException.class, ()->{  
 address1 = new Address("1a",777, "", "v7n2m8", "surrey");  
 });  
 *assertTrue*(ex.getMessage().equals("Invalid street name: "));  
  
 ex = *assertThrows*(IllegalArgumentException.class, ()->{  
 address1 = new Address("1a",777, "abcdefghijklmnopqrstu", "v7n2m8", "surrey");  
 });  
 *assertTrue*(ex.getMessage().equals("Invalid street name: abcdefghijklmnopqrstu"));  
 }  
  
 @Test  
 void getExpectedExceptionsPostalCode()  
 {  
 NullPointerException ex2 = *assertThrows*(NullPointerException.class, ()->{  
 address1 = new Address("1a",777, "56th avenue", null, "surrey");  
 });  
 *assertTrue*(ex2.getMessage().equals("Invalid postal code: null"));  
  
 IllegalArgumentException ex = *assertThrows*(IllegalArgumentException.class, ()->{  
 address1 = new Address("1a",777, "56th avenue", "1234", "surrey");  
 });  
 *assertTrue*(ex.getMessage().equals("Invalid postal code: 1234"));  
  
 ex = *assertThrows*(IllegalArgumentException.class, ()->{  
 address1 = new Address("1a",777, "56th avenue", "1234567", "surrey");  
 });  
 *assertTrue*(ex.getMessage().equals("Invalid postal code: 1234567"));  
 }  
  
 @Test  
 void getExpectedExceptionsCity()  
 {  
 NullPointerException ex2 = *assertThrows*(NullPointerException.class, ()->{  
 address1 = new Address("1a",777, "56th avenue", "v7n2m8", null);  
 });  
 *assertTrue*(ex2.getMessage().equals("Invalid city: null"));  
  
 IllegalArgumentException ex = *assertThrows*(IllegalArgumentException.class, ()->{  
 address1 = new Address("1a",777, "56th avenue", "v7n2m8", "");  
 });  
 *assertTrue*(ex.getMessage().equals("Invalid city: "));  
  
 ex = *assertThrows*(IllegalArgumentException.class, ()->{  
 address1 = new Address("1a",777, "56th avenue", "v7n2m8", "0123456789012345678901234567890");  
 });  
 *assertTrue*(ex.getMessage().equals("Invalid city: 0123456789012345678901234567890"));  
 }  
}

import org.junit.jupiter.api.AfterEach;  
import org.junit.jupiter.api.BeforeEach;  
import org.junit.jupiter.api.Test;  
  
import static org.junit.jupiter.api.Assertions.\*;  
  
class PropertyTest  
{  
 private Property property1;  
 private Property property2;  
 private Property property3;  
 private Address address;  
  
 @BeforeEach  
 void setUp()  
 {  
 address = new Address("1a", 777, "56th avenue", "v7n2m8", "surrey");  
  
 property1 = new Property(499000, address, 1, true, "resiDEnce", "11111");  
 property2 = new Property(1000000, address, 2, false, "commErcial", "22222");  
 property3 = new Property(840000, address, 3, false, "rETAil", "xxxxx");  
 }  
  
 @AfterEach  
 void tearDown()  
 {  
 address = null;  
  
 property1 = null;  
 property2 = null;  
 property3 = null;  
 }  
  
 @Test  
 void getPriceUsd()  
 {  
 *assertEquals*(499000, property1.getPriceUsd());  
 *assertEquals*(1000000, property2.getPriceUsd());  
 *assertEquals*(840000, property3.getPriceUsd());  
 }  
  
 @Test  
 void getAddress()  
 {  
 *assertEquals*(address, property1.getAddress());  
 }  
  
 @Test  
 void getNumberOfBedrooms()  
 {  
 *assertEquals*(1, property1.getNumberOfBedrooms());  
 *assertEquals*(2, property2.getNumberOfBedrooms());  
 *assertEquals*(3, property3.getNumberOfBedrooms());  
 }  
  
 @Test  
 void hasSwimmingPool()  
 {  
 *assertTrue*(property1.hasSwimmingPool());  
 *assertFalse*(property2.hasSwimmingPool());  
 *assertFalse*(property3.hasSwimmingPool());  
 }  
  
 @Test  
 void getType()  
 {  
 *assertEquals*("resiDEnce", property1.getType());  
 *assertEquals*("commErcial", property2.getType());  
 *assertEquals*("rETAil", property3.getType());  
 }  
  
 @Test  
 void getPropertyId()  
 {  
 *assertEquals*("11111", property1.getPropertyId());  
 *assertEquals*("22222", property2.getPropertyId());  
 *assertEquals*("xxxxx", property3.getPropertyId());  
 }  
  
 @Test  
 void setPriceUsd()  
 {  
 property1.setPriceUsd(777123);  
 *assertEquals*(777123, property1.getPriceUsd());  
 }  
  
  
 @Test  
 void getExpectedExceptionsPriceUsd()  
 {  
 IllegalArgumentException ex = *assertThrows*(IllegalArgumentException.class, ()->{  
 property1 = new Property(-0.01, address, 1, true, "residence", "11111");  
 });  
 *assertTrue*(ex.getMessage().equals("Invalid price: -0.01"));  
 }  
  
 @Test  
 void getExpectedExceptionsAddress()  
 {  
 NullPointerException ex = *assertThrows*(NullPointerException.class, ()->{  
 property1 = new Property(499000, null, 1, true, "residence", "11111");  
 });  
 *assertTrue*(ex.getMessage().equals("Invalid address: null"));  
 }  
  
 @Test  
 void getExpectedExceptionsNumBedrooms()  
 {  
 IllegalArgumentException ex = *assertThrows*(IllegalArgumentException.class, ()->{  
 property1 = new Property(499000, address, 0, true, "residence", "11111");  
 });  
 *assertTrue*(ex.getMessage().equals("Invalid number of bedrooms: 0"));  
  
 ex = *assertThrows*(IllegalArgumentException.class, ()->{  
 property1 = new Property(499000, address, 21, true, "residence", "11111");  
 });  
 *assertTrue*(ex.getMessage().equals("Invalid number of bedrooms: 21"));  
 }  
  
 @Test  
 void getExpectedExceptionsPropertyType()  
 {  
 NullPointerException ex = *assertThrows*(NullPointerException.class, ()->{  
 property1 = new Property(499000, address, 1, true, null, "11111");  
 });  
 *assertTrue*(ex.getMessage().equals("Invalid property type: null"));  
  
 IllegalArgumentException ex2 = *assertThrows*(IllegalArgumentException.class, ()->{  
 property1 = new Property(499000, address, 2, true, "residencee", "11111");  
 });  
 *assertTrue*(ex2.getMessage().equals("Invalid property type: residencee"));  
 }  
  
 @Test  
 void getExpectedExceptionsPropertyId()  
 {  
 NullPointerException ex = *assertThrows*(NullPointerException.class, ()->{  
 property1 = new Property(499000, address, 1, true, "residence", null);  
 });  
 *assertTrue*(ex.getMessage().equals("Invalid property id: null"));  
  
  
 IllegalArgumentException ex2 = *assertThrows*(IllegalArgumentException.class, ()->{  
 property1 = new Property(499000, address, 1, true, "residence", "");  
 });  
 *assertTrue*(ex2.getMessage().equals("Invalid property id: "));  
  
 ex2 = *assertThrows*(IllegalArgumentException.class, ()->{  
 property1 = new Property(499000, address, 1, true, "residence", "1234567");  
 });  
 *assertTrue*(ex2.getMessage().equals("Invalid property id: 1234567"));  
 }  
}

import org.junit.jupiter.api.AfterEach;  
import org.junit.jupiter.api.BeforeEach;  
import org.junit.jupiter.api.Test;  
  
import java.util.ArrayList;  
import java.util.HashMap;  
import java.util.Set;  
  
import static org.junit.jupiter.api.Assertions.\*;  
  
class AgencyTest  
{  
 private Agency agency;  
  
 private Address a1 = new Address("1a", 777, "56th avenue", "v7n2m8", "surrey");  
 private Property p1 = new Property(499000.00, a1, 2, false, "residence", "abc123");  
  
 private Address a2 = new Address(null, 123, "main street", "v7r2g2", "west vancouver");  
 private Property p2 = new Property(5999999.00, a2, 5, true, "residence", "xyz789");  
  
 private Address a3 = new Address(null, 456, "elm street", "90210", "los angeles");  
 private Property p3 = new Property(2500000.00, a3, 6, true, "residence", "777def");  
  
 private Address a4 = new Address("44", 1111, "maple street", "v8y3r5", "vancouver");  
 private Property p4 = new Property(1000000.00, a4, 1, false, "retail", "876tru");  
  
 private Address a5 = new Address("9", 99, "gretzky way", "t6v7h3", "toronto");  
 private Property p5 = new Property(99999.00, a5, 1, false, "commercial", "9999");  
  
 private Address a6 = new Address("b", 711, "country road", "v8h5f5", "maple ridge");  
 private Property p6 = new Property(740100.00, a6, 3, false, "residence", "mr6789");  
  
 private Address a7 = new Address(null, 8785, "pinnacle avenue", "v9u3h3", "north vancouver");  
 private Property p7 = new Property(15000000.00, a7, 20, true, "residence", "78444a");  
  
 private Address a8 = new Address(null, 800, "elm street", "90557", "los angeles");  
 private Property p8 = new Property(7100000.00, a8, 10, false, "residence", "mmm33");  
  
 private Address a9 = new Address(null, 1515,"main street", "v8y7r3", "west vancouver");  
 private Property p9 = new Property(4000000.00, a9, 2, true, "commercial", "678T");  
  
 private Address a10 = new Address("6", 60, "60th street", "v8u9b1", "burnaby");  
 private Property p10 = new Property(700000.00, a10, 2, true, "retail", "y6yyy");  
  
 private Address a11 = new Address("7h", 1500, "railway avenue", "v9v5v4", "richmond");  
 private Property p11 = new Property(840000.00, a11, 4, false, "commercial", "A1212");  
  
 private Address a12 = new Address(null, 333, "elm street", "90111", "los angeles");  
 private Property p12 = new Property(1600000.00, a12, 3, false, "residence", "9000a");  
  
  
 @BeforeEach  
 void setUp()  
 {  
 agency = new Agency("BCIT Ltd");  
  
  
  
  
  
 agency.addProperty(p1);  
 agency.addProperty(p2);  
 agency.addProperty(p3);  
 agency.addProperty(p4);  
 agency.addProperty(p5);  
 agency.addProperty(p6);  
 agency.addProperty(p7);  
 agency.addProperty(p8);  
 agency.addProperty(p9);  
 agency.addProperty(p10);  
 agency.addProperty(p11);  
 agency.addProperty(p12);  
  
 }  
  
  
 @AfterEach  
 void tearDown()  
 {  
 agency = null;  
 }  
  
  
  
  
 @Test  
 void addGetProperty()  
 {  
 *assertNull*(agency.getProperty("x"));  
  
 Address a13 = new Address(null, 333, "elm street", "90111", "los angeles");  
 Property p13 = new Property(1600000.00, a13, 3, false, "residence", "x");  
  
 agency.addProperty(p13);  
  
 *assertSame*(p13, agency.getProperty("x"));  
 }  
  
  
 @Test  
 void removeProperty()  
 {  
 *assertNull*(agency.getProperty("x"));  
  
 Address a13 = new Address(null, 333, "elm street", "90111", "los angeles");  
 Property p13 = new Property(1600000.00, a13, 3, false, "residence", "x");  
  
 agency.addProperty(p13);  
  
 *assertSame*(p13, agency.getProperty("x"));  
  
 agency.removeProperty("x");  
  
 *assertNull*(agency.getProperty("x"));  
  
 }  
  
 @Test  
 void getTotalPropertyValues()  
 {  
 *assertEquals*(40079098, agency.getTotalPropertyValues());  
 }  
  
 @Test  
 void getPropertiesWithPools()  
 {  
 ArrayList<Property> propertiesWithPools = new ArrayList<>();  
 propertiesWithPools.add(p2);  
 propertiesWithPools.add(p3);  
 propertiesWithPools.add(p7);  
 propertiesWithPools.add(p9);  
 propertiesWithPools.add(p10);  
  
  
  
  
 *assertTrue*(propertiesWithPools.size() == agency.getPropertiesWithPools().size() &&  
 propertiesWithPools.containsAll(agency.getPropertiesWithPools()) &&  
 agency.getPropertiesWithPools().containsAll(propertiesWithPools));  
  
  
  
 }  
  
 @Test  
 void getPropertiesBetween()  
 {  
 Property[] matches = new Property[5];  
 Property[] agencyResults = agency.getPropertiesBetween(700001, 2500000);  
  
 *assertTrue*(agencyResults.length == 5);  
  
 matches[0] = p3;  
 matches[1] = p4;  
 matches[2] = p6;  
 matches[3] = p11;  
 matches[4] = p12;  
  
 boolean found0 = false;  
 boolean found1 = false;  
 boolean found2 = false;  
 boolean found3 = false;  
 boolean found4 = false;  
  
  
  
 for(Property property: agencyResults)  
 {  
 if(property.getPropertyId().equalsIgnoreCase(matches[0].getPropertyId()))  
 {  
 found0 = true;  
 }  
  
 if(property.getPropertyId().equalsIgnoreCase(matches[1].getPropertyId()))  
 {  
 found1 = true;  
 }  
  
 if(property.getPropertyId().equalsIgnoreCase(matches[2].getPropertyId()))  
 {  
 found2 = true;  
 }  
  
 if(property.getPropertyId().equalsIgnoreCase(matches[3].getPropertyId()))  
 {  
 found3 = true;  
 }  
  
 if(property.getPropertyId().equalsIgnoreCase(matches[4].getPropertyId()))  
 {  
 found4 = true;  
 }  
 }  
  
 *assertTrue*(found0 && found1 && found2 && found3 && found4);  
  
  
  
 }  
  
 @Test  
 void getPropertiesOn()  
 {  
 ArrayList<Address> propertiesOnElmStreet = new ArrayList<>();  
 propertiesOnElmStreet.add(p3.getAddress());  
 propertiesOnElmStreet.add(p8.getAddress());  
 propertiesOnElmStreet.add(p12.getAddress());  
  
  
 *assertTrue*(propertiesOnElmStreet.size() == agency.getPropertiesOn("elm street").size() &&  
 propertiesOnElmStreet.containsAll(agency.getPropertiesOn("elm street")) &&  
 agency.getPropertiesOn("elm street").containsAll(propertiesOnElmStreet));  
  
 *assertNull*(agency.getPropertiesOn("fake street"));  
  
 }  
  
  
  
 @Test  
 void getPropertiesWithBedrooms()  
 {  
 HashMap<String, Property> expectedMatches = new HashMap<>();  
  
 expectedMatches.put(p2.getPropertyId(), p2);  
 expectedMatches.put(p3.getPropertyId(), p3);  
 expectedMatches.put(p8.getPropertyId(), p8);  
 expectedMatches.put(p11.getPropertyId(), p11);  
  
 HashMap<String, Property> fromAgency = agency.getPropertiesWithBedrooms(4,12);  
  
 *assertEquals*(expectedMatches.size(), fromAgency.size());  
  
  
 Set<String> propertyIds = expectedMatches.keySet();  
  
 // are all four expected properties in the agency?  
 for(String expectedPropertyId: propertyIds)  
 {  
 *assertTrue*(fromAgency.containsKey(expectedPropertyId));  
 }  
  
  
  
 *assertNull*(agency.getPropertiesWithBedrooms(7, 9));  
  
  
  
  
 }  
  
 @Test  
 void getPropertiesOfType()  
 {  
  
 ArrayList<String> agencyData = agency.getPropertiesOfType("commerCIAl");  
  
 boolean s1found = false;  
 boolean s2found = false;  
 boolean s3found = false;  
 boolean s4found = false;  
  
 String s1 = "Type: COMMERCIAL\n";  
 String s2 = ") Property 9999: unit #9 at 99 Gretzky Way T6V7H3 in Toronto (1 bedroom): $99999.\n";  
 String s3 = ") Property 678T: 1515 Main Street V8Y7R3 in West Vancouver (2 bedrooms plus pool): $4000000.\n";  
 String s4 = ") Property A1212: unit #7h at 1500 Railway Avenue V9V5V4 in Richmond (4 bedrooms): $840000.\n";  
  
 // order doesn't matter, so the numbers don't matter, but the strings must be present  
 for(String string: agencyData)  
 {  
 if(string.contains(s1))  
 {  
 s1found = true;  
  
 }  
 if(string.contains(s2))  
 {  
 s2found = true;  
  
 }  
 if(string.contains(s3))  
 {  
 s3found = true;  
  
 }  
 if(string.contains(s4))  
 {  
 s4found = true;  
  
 }  
 }  
 *assertEquals*(4, agencyData.size());  
 *assertTrue*(s1found && s2found && s3found && s4found);  
  
 // new check, for property type that does not exist  
  
 boolean s5found = false;  
 boolean s6found = false;  
  
 ArrayList<String> agencyData2 = agency.getPropertiesOfType("fake fake fake");  
 // order doesn't matter, so the numbers don't matter, but the strings must be present  
  
 *assertEquals*(2, agencyData2.size());  
 for(String string2: agencyData2)  
 {  
 if (string2.contains("Type: FAKE FAKE FAKE"))  
 {  
 s5found = true;  
  
 }  
 if (string2.contains("<none found>"))  
 {  
 s6found = true;  
 }  
 }  
 *assertTrue*(s5found && s6found);  
  
 }  
}