Task [1]

protected String programmeName;

public String getProgrammeName() {

return programmeName;

}

public void setProgrammeName(String programmeName) {

this.programmeName = programmeName;

}

Task [2]

Constructor:

public Student(String fn, int registrationNumber, String email) {

this.fullName = fn;

this.registrationNumber = registrationNumber;

this.email = email;

this.numberModules = 0;

this.programmeName = "Bachelor of Science";

}

Second Constructor:

public Student(String fullName, int registrationNumber, String email, String programmeName) {

this.fullName = fullName;

this.registrationNumber = registrationNumber;

this.email = email;

this.programmeName = programmeName;

}

Task [3]

protected boolean isARUAA;

public Student(String fn, int registrationNumber, String email) {

this.fullName = fn;

this.registrationNumber = registrationNumber;

this.email = email;

this.numberModules = 0;

this.programmeName = "Bachelor of Science";

this.isARUAA = false;

}

public boolean isARUAA() {

return isARUAA;

}

Task [4]

public void isARUAAStudent() {

this.isARUAA = true;

}

public void isNotARUAAStudent() {

this.isARUAA = false;

}

Task [5]

public boolean addModule(int inter) {

boolean isAltered = false;

if (numberModules < MAX\_NUM\_MODULES)

{

this.numberModules = this.numberModules + inter;

isAltered = true;

}

return isAltered;

}

Task [6]

public class MScStudent extends Student

{

protected String researchTitle;

protected Professor supervisor;

}

Task [7]

public MScStudent(String fn, int registrationNumber, String email, String researchTitle, Professor supervisor) {

super(fn, registrationNumber, email);

this.researchTitle = researchTitle;

this.supervisor = supervisor;

}

Task [8]

public String toString() {

return super.toString() + " researchTitle=" + researchTitle + ", supervisor=" + supervisor;

}

Task [9]

if (n.left() !=null) inOrder(n.left(), sb);

sb.append(n.getPersonName() + ",");

if (n.right() !=null) inOrder(n.right(), sb);

Task [10]

while (n != null) {

if (name.compareTo(n.getPersonName()) == 0) {

return n;

}

// This will drop out of the loop naturally if there's no appropriate subnode

n = 1 < name.compareTo(n.getPersonName()) ? n.left() : n.right();

}

return null;

Task [11]

ArrayList<Module> sorted = **new** ArrayList<Module>();

// **TODO**

sorted.addAll(*merge*(list));

**if** (attr == "name"&& ascending == **true**) {

sorted.sort(Comparator.*comparing*(Module::getName));

}

**if** (attr == "name"&& ascending == **false**) {

sorted.sort(Comparator.*comparing*(Module::getName).reversed());

}

**if** (attr == "code"&& ascending == **true**) {

sorted.sort(Comparator.*comparing*(Module::getCode));

}

**if** (attr == "code" && ascending == **false**) {

sorted.sort(Comparator.*comparing*(Module::getCode).reversed());

}

**return** sorted;

}

**private** **static** ArrayList<Module> merge(ArrayList<Module> a) {

**if** (a.size()<=1) **return** a; // small list don't need to be merged

// SEPARATE

**int** mid = a.size()/2; // estimate half the size

ArrayList<Module> left = **new** ArrayList<Module>();

ArrayList<Module> right = **new** ArrayList<Module>();

**for**(**int** i = 0; i < mid; i++) left.add(a.remove(0)); // put first half part in left

**while** (a.size()!=0) right.add(a.remove(0)); // put the remainings in right

// Here a is now empty

// MERGE PARTS INDEPENDANTLY

*merge*(left); // merge the left part

*merge*(right); // merge the right part

// MERGE PARTS

// while there is something in the two lists

**while** (left.size()!=0 && right.size()!=0) {

// compare both heads, add the lesser into the result and remove it from its list

**if** (left.get(0).compareTo(right.get(0))<0) a.add(left.remove(0));

**else** a.add(right.remove(0));

}

// fill the result with what remains in left OR right (both can't contains elements)

**while**(left.size()!=0) a.add(left.remove(0));

**while**(right.size()!=0) a.add(right.remove(0));

**return** a;

}

}

Task [12]

**protected** Module binarySearch(ArrayList<Module> list, String name)

{

// Use the binary search algorithm to

// Initially this method just returns null - you need to fix this.

// **TODO**

**for** (Module v : modules)

{

**if**(v.getName() == name) {

**return** v;

}

}

**return** **null**;

}

Task [13]

* inOrderTest()

assertEquals(stirling.inWalk(0), "John J,Alma A,Kate K,Loius L,");

* binarySearchTest()

assertEquals(stirling.binarySearch("Algorithms 3").toString(),"Module: 8764 - Algorithms 3 -- Students: Borba B,Jess Z,Hugh J,Jess Z,");

assertEquals(stirling.binarySearch("Biology 1"), null);

End.