```
1   // EXAM #3 Sample Solution Student Wrapper
2   // Instructions
3   // ***************************************************************
4   /* Complete the definition of the following MUTABLE class named StudentWrapper.  This
    class contains a set of nested classes and interface that can be used to represent
    students in some school.  Some of these classes are poorly designed and you job is to
    improve this design.
5
6   Your job consist of completing the following tasks:
7
8   1-Complete the definition of the AbstractStudent abstract class to gather all common
    elements in the CIICStudent and INSOStudent classes.
9   2-Refactor the CIICStudent and INSOStudent classes  to remove duplicate code now
    collected in the AbstractStudent class.
10  3-Complete the definition of the IncomeTaxPayer interface supporting the obligation of
    a student piece to pay taxes.
11  4-Complete the definition of the getIncomeTaxRate() instance method in the appropriate
    class to implement the IncomeTaxPayer interface.
12  5-Complete the definitions of the getSpecialtyGPA abstract instance method in the
    INSOStudent class.
13  6-Complete the definition of the clone() method in the CIICStudent class to implement
    the Cloneable interface.
14  7-Complete the definition of the findINSOStudent instance method in the INSOStudent
    class. YOUR METHOD MUST BE RECURSIVE.
15  8-Complete the definition of the getTopCIICStudent static method in the CIICStudent
    class. YOUR METHOD MUST BE RECURSIVE.
16  */
17
18  // Start File / Given Code
19  // ***************************************************************
20  public class StudentWrapperStarter { // Change / Rename to StudentWrapper
21
22      public enum Gender {
23          MALE,
24          FEMALE
25      }
26
27      public enum Merit {
28          LAUDE,
29          CUM_LAUDE,
30          MAGNA_CUM_LAUDE
31      }
32
33      /**
34       * Exercise #3
35       * An interface that represents an Income Tax Payer.
36       * Contains the method getIncomeTaxRate.
37       *
38       */
39      public interface IncomeTaxPayer {
40          // YOUR CODE HERE
41      }
42
43      /**
44       * Exercise #1
45       *
46       * An abstract class that holds the properties and implements the methods that are
         common
47       * to students of both specialties CIIC and INSO. All the moved properties must
         remain
48       * PRIVATE and the methods PUBLIC.
49       *
50       * The abstract class should supply an appropriate constructor to be used by
51       * subclass constructors to initialize the private properties in the abstract class.
52       * Any variables that do not have the same definition should not be moved from their
53       * respective classes (Like INSOGpa).
```

```
54          * Any methods that have are the same, but have different implementations or use
            different
55          * variables can be moved to the abstract clas, but only its declaration, not the
            implementation.
56          *
57          */
58         public static abstract class AbstractStudent implements Cloneable, IncomeTaxPayer {
59
60             // YOUR CODE HERE
61
62         }
63
64         /**
65          * Exercise #2 (Part A)
66          * Refactor this class to remove any property or method that was
67          * moved to the AbstractStudent class.  You should modify the
68          * constructor to call the super constructor appropriately.
69          *
70          * A class that represents a CIIC Student
71          *
72          */
73         public static class CIICStudent extends AbstractStudent {
74
75             private String idNumber;
76             private String firstName;
77             private String lastName;
78             private Gender gender;
79             private double GPA;
80             private double ciicGPA;
81             private int    ciicCredits;
82
83             public CIICStudent(String idNumber, String firstName, String lastName, Gender
               gender, double GPA, double ciicGPA, int ciicCredits) {
84                 super();
85                 this.idNumber = idNumber;
86                 this.firstName = firstName;
87                 this.lastName = lastName;
88                 this.gender = gender;
89                 this.GPA = GPA;
90                 this.ciicGPA = ciicGPA;
91                 this.ciicCredits = ciicCredits;
92             }
93
94             public String getIdNumber() {
95                 return idNumber;
96             }
97             public void setIdNumber(String idNumber) {
98                 this.idNumber = idNumber;
99             }
100            public String getFirstName() {
101                return firstName;
102            }
103            public void setFirstName(String firstName) {
104                this.firstName = firstName;
105            }
106            public String getLastName() {
107                return lastName;
108            }
109            public void setLastName(String lastName) {
110                this.lastName = lastName;
111            }
112            public Gender getGender() {
113                return gender;
114            }
115            public void setGender(Gender gender) {
116                this.gender = gender;
```

```
117         }
118         public double getGPA() {
119             return GPA;
120         }
121         public void setGPA(double GPA) {
122             this.GPA = GPA;
123         }
124 
125         public double getCIICGPA() {
126             return ciicGPA;
127         }
128 
129         public int getCIICCredits() {
130             return ciicCredits;
131         }
132 
133         public double getSpecialtyGPA() {
134             return ciicGPA;
135         }
136 
137         /**
138          * Exercise #6
139          * Returns a deep clone of the target object. A deep clone is a copy of
140          * the object replacing any references to internal objects with references
141          * to deep clones of these internal objects.
142          * Implements Cloneable interface
143          */
144         @Override
145         public Object clone() {
146             // YOUR CODE HERE
147             return null; // Dummy return
148         }
149 
150         public static CIICStudent maxGPA(CIICStudent s1, CIICStudent s2) {
151             return (s1.getCIICGPA()>s2.getCIICGPA() ? s1 : s2);
152         }
153 
154         /**
155          * Exercise #8
156          * A method that returns the first student with the highest CIIC GPA
157          * The method MUST BE RECURSIVE.
158          * @param start
159          * @param students
160          * @return
161          */
162         public static CIICStudent getTopCIICStudent(int start, CIICStudent[] students) {
163             // YOUR CODE HERE
164             return null; // Dummy return
165         }
166 
167         public String toString() {
168             return "Student: " + this.getIdNumber();
169         }
170 
171 
172         /**
173          * Exercise #4
174          * A method that returns the students tax rate
175          * Hint: Student have 5% tax rate.
176          */
177 
178         public double getIncomeTaxRate() {
179             return 0; // Dummy return
180         }
181 
182     }
```

```
183
184         /**
185          * Exercise #2 (Part B)
186          * Refactor this class to remove any property or method that was
187          * moved to the AbstractStudent class.  You should modify the
188          * constructor to call the super constructor appropriately.
189          *
190          * A class that represents a INSO Student
191          *
192          */
193         public static class INSOStudent extends AbstractStudent {
194
195             private String idNumber;
196             private String firstName;
197             private String lastName;
198             private Gender gender;
199             private double GPA;
200             private double insoGPA;
201             private int    insoCredits;
202
203             public INSOStudent(String idNumber, String firstName, String lastName, Gender
                gender, double GPA, double insoGPA, int insoCredits) {
204                 super();
205                 this.idNumber = idNumber;
206                 this.firstName = firstName;
207                 this.lastName = lastName;
208                 this.gender = gender;
209                 this.GPA = GPA;
210                 this.insoGPA = insoGPA;
211                 this.insoCredits = insoCredits;
212             }
213
214             public String getIdNumber() {
215                 return idNumber;
216             }
217             public void setIdNumber(String idNumber) {
218                 this.idNumber = idNumber;
219             }
220             public String getFirstName() {
221                 return firstName;
222             }
223             public void setFirstName(String firstName) {
224                 this.firstName = firstName;
225             }
226             public String getLastName() {
227                 return lastName;
228             }
229             public void setLastName(String lastName) {
230                 this.lastName = lastName;
231             }
232             public Gender getGender() {
233                 return gender;
234             }
235             public void setGender(Gender gender) {
236                 this.gender = gender;
237             }
238             public double getGPA() {
239                 return GPA;
240             }
241             public void setGPA(double GPA) {
242                 this.GPA = GPA;
243             }
244
245             public double getINSOGPA() {
246                 return insoGPA;
247             }
```

```
248
249             public int getINSOCredits() {
250                 return insoCredits;
251             }
252
253             @Override
254             public Object clone() {
255                 // Do not implement
256                 return null;
257             }
258
259             /**
260              * Exercise #4
261              * A method that returns the students tax rate
262              * Hint: Student have 5% tax rate.
263              */
264
265             public double getIncomeTaxRate() {
266                 return 0;
267             }
268
269             /**
270              * Exercise #7
271              * A method that returns the position of the target INSOStudent in the given
                 list.
272              * The method MUST BE RECURSIVE
273              * @param list
274              * @param start
275              * @return
276              */
277             public boolean findINSOStudent(INSOStudent[] list, int start) {
278                 // YOUR CODE HERE
279                 return false; // Dummy return
280             }
281
282             /**
283              * Exercise #5
284              * A method that returns the student specialty GPA
285              * @return
286              */
287             public double getSpecialtyGPA() {
288                 // YOUR CODE HERE
289                 return 0; // Dummy return
290             }
291         }
292     }
293
294     // Sample Solution
295     // ********************************************************************
296     public class StudentWrapper {
297
298         public enum Gender {
299             MALE,
300             FEMALE
301         }
302
303         public enum Merit {
304             LAUDE,
305             CUM_LAUDE,
306             MAGNA_CUM_LAUDE
307         }
308
309         public interface IncomeTaxPayer {
310             double getIncomeTaxRate();
311         }
312
```

```
313     public static abstract class AbstractStudent implements Cloneable, IncomeTaxPayer{
314
315         private String idNumber;
316         private String firstName;
317         private String lastName;
318         private Gender gender;
319         private double GPA;
320
321         public AbstractStudent(String idNumber, String firstName, String lastName,
            Gender gender, double GPA) {
322             super();
323             this.idNumber = idNumber;
324             this.firstName = firstName;
325             this.lastName = lastName;
326             this.gender = gender;
327             this.GPA = GPA;
328         }
329
330         public String getIdNumber() {
331             return idNumber;
332         }
333         public void setIdNumber(String idNumber) {
334             this.idNumber = idNumber;
335         }
336         public String getFirstName() {
337             return firstName;
338         }
339         public void setFirstName(String firstName) {
340             this.firstName = firstName;
341         }
342         public String getLastName() {
343             return lastName;
344         }
345         public void setLastName(String lastName) {
346             this.lastName = lastName;
347         }
348         public Gender getGender() {
349             return gender;
350         }
351         public void setGender(Gender gender) {
352             this.gender = gender;
353         }
354         public double getGPA() {
355             return GPA;
356         }
357         public void setGPA(double GPA) {
358             this.GPA = GPA;
359         }
360         public double getIncomeTaxRate() {
361             return 0.05;
362         }
363
364         public abstract double getSpecialtyGPA();
365     }
366
367
368     public static class CIICStudent extends AbstractStudent {
369
370         private double ciicGPA;
371         private int    ciicCredits;
372
373         public CIICStudent(String idNumber, String firstName, String lastName, Gender
            gender, double GPA, double ciicGPA, int ciicCredits) {
374             super(idNumber, firstName, lastName, gender, GPA);
375             this.ciicGPA = ciicGPA;
376             this.ciicCredits = ciicCredits;
```

```
377             }
378
379             public double getCIICGPA() {
380                 return ciicGPA;
381             }
382
383             public int getCIICCredits() {
384                 return ciicCredits;
385             }
386
387             public double getSpecialtyGPA() {
388                 return ciicGPA;
389             }
390
391             @Override
392             public Object clone() {
393                 CIICStudent newCIICStudent = new CIICStudent(
394                         new String(this.getIdNumber()),
395                         new String(this.getFirstName()),
396                         new String(this.getLastName()),
397                         this.getGender(),
398                         this.getGPA(),
399                         this.ciicGPA,
400                         this.ciicCredits);
401                 return newCIICStudent;
402             }
403
404             public static CIICStudent maxGPA(CIICStudent s1, CIICStudent s2) {
405                 return (s1.getCIICGPA()>s2.getCIICGPA() ? s1 : s2);
406             }
407
408             public static CIICStudent getTopCIICStudent(int start, CIICStudent[] students) {
409                 if (students.length == 0) {
410                     return null;
411                 }
412                 else if (start == students.length-1) {
413                     return students[students.length-1];
414                 }
415                 else {
416                     return maxGPA(students[start], getTopCIICStudent(start + 1, students));
417                 }
418             }
419
420         }
421
422         public static class INSOStudent extends AbstractStudent {
423
424             private double insoGPA;
425             private int    insoCredits;
426
427             public INSOStudent(String idNumber, String firstName, String lastName, Gender
                gender, double GPA, double insoGPA, int insoCredits) {
428                 super(idNumber, firstName, lastName, gender, GPA);
429                 this.insoGPA = insoGPA;
430                 this.insoCredits = insoCredits;
431             }
432
433             public double getINSOGPA() {
434                 return insoGPA;
435             }
436
437             public int getINSOCredits() {
438                 return insoCredits;
439             }
440
441             public double getSpecialtyGPA() {
```

```
442             return insoGPA;
443         }
444
445         @Override
446         public Object clone() {
447             INSOStudent newINSOStudent = new INSOStudent(
448                     new String(this.getIdNumber()),
449                     new String(this.getFirstName()),
450                     new String(this.getLastName()),
451                     this.getGender(),
452                     this.getGPA(),
453                     this.insoGPA,
454                     this.insoCredits);
455             return newINSOStudent;
456         }
457
458         public boolean findINSOStudent(INSOStudent[] list, int start) {
459             if (start >= list.length) {
460                 return false;
461             }
462             else if (list[start].getIdNumber().equals(this.getIdNumber())) {
463                     return true;
464             }
465             else {
466                 return findINSOStudent(list, start + 1);
467             }
468         }
469     }
470 }
471
472
473 // Tests
474 // ****************************************************************
475 import static org.junit.Assert.assertEquals;
476 import static org.junit.Assert.assertFalse;
477 import static org.junit.Assert.assertTrue;
478 import static org.junit.Assert.fail;
479
480 import java.lang.reflect.Method;
481 import java.util.Arrays;
482
483 import org.junit.Before;
484 import org.junit.Rule;
485 import org.junit.Test;
486 import org.junit.rules.ExpectedException;
487
488 public class StudentWrapperTester {
489
490     public StudentWrapper.CIICStudent ciic1;
491     public StudentWrapper.CIICStudent ciic2;
492     public StudentWrapper.CIICStudent ciic3;
493     public StudentWrapper.CIICStudent ciic4;
494
495     public StudentWrapper.INSOStudent inso1;
496     public StudentWrapper.INSOStudent inso2;
497     public StudentWrapper.INSOStudent inso3;
498     public StudentWrapper.INSOStudent inso4;
499
500     StudentWrapper.CIICStudent[] emptyCIICList;
501     StudentWrapper.CIICStudent[] allCIICList;
502     StudentWrapper.CIICStudent[] hugeCIICList;
503
504     StudentWrapper.INSOStudent[] emptyINSOList;
505     StudentWrapper.INSOStudent[] allINSOList;
506     StudentWrapper.INSOStudent[] hugeINSOList;
507
```

```
508         @Before
509         public void setUp() {
510             ciic1 = new StudentWrapper.CIICStudent("802-14-1234", "Jose", "Santiago",
                StudentWrapper.Gender.MALE, 3.67, 3.5, 21);
511             ciic2 = new StudentWrapper.CIICStudent("802-14-1111", "Pedro", "Martinez",
                StudentWrapper.Gender.MALE, 2.89, 3.1, 45);
512             ciic3 = new StudentWrapper.CIICStudent("802-14-3333", "Juan", "Velez",
                StudentWrapper.Gender.MALE, 3.5, 4.0, 120);
513             ciic4 = new StudentWrapper.CIICStudent("802-14-4444", "Marta", "Ramirez",
                StudentWrapper.Gender.FEMALE, 4.0, 3.5, 155);
514             inso1 = new StudentWrapper.INSOStudent("802-14-1235", "Maria", "Santiago",
                StudentWrapper.Gender.FEMALE, 3.89, 3.2, 30);
515             inso2 = new StudentWrapper.INSOStudent("802-14-2222", "Ana", "Lopez",
                StudentWrapper.Gender.FEMALE, 3.23, 3.5, 60);
516             inso3 = new StudentWrapper.INSOStudent("802-14-1235", "Joe", "Doe",
                StudentWrapper.Gender.MALE, 3.89, 3.2, 30);
517             inso4 = new StudentWrapper.INSOStudent("802-14-2222", "John", "Doe",
                StudentWrapper.Gender.MALE, 3.23, 3.5, 60);
518
519             emptyCIICList = new StudentWrapper.CIICStudent[] {};
520             allCIICList = new StudentWrapper.CIICStudent[] {ciic1, ciic2, ciic3, ciic4};
521             hugeCIICList = new StudentWrapper.CIICStudent[10000];
522             Arrays.fill(hugeCIICList, ciic1);
523
524             emptyINSOList = new StudentWrapper.INSOStudent[] {};
525             allINSOList = new StudentWrapper.INSOStudent[] {inso1, inso2, inso3, inso4};
526             hugeINSOList = new StudentWrapper.INSOStudent[100000];
527             Arrays.fill(hugeINSOList, inso2);
528
529         }
530
531         @Test
532         public void testStudentAbstract() {
533             assertTrue("CIIC Student should be instance of the AbstractStudent Class", ciic1
                 instanceof StudentWrapper.AbstractStudent);
534             assertTrue("INSO Student should be instance of the AbstractStudent Class", inso1
                 instanceof StudentWrapper.AbstractStudent);
535
536             StudentWrapper.AbstractStudent absCIIC1 = (StudentWrapper.AbstractStudent) ciic1;
537             StudentWrapper.AbstractStudent absINSO1 = (StudentWrapper.AbstractStudent) inso1;
538
539             boolean hasMethods = true;
540             try {
541                 Class<?> c = Class.forName("StudentWrapper");
542                 Class<?> cl[] = c.getDeclaredClasses();
543                 for(Class<?> cls : cl) {
544                     if(cls.toString().contains("AbstractStudent")) {
545                         Method[] methods = cls.getDeclaredMethods();
546                         assertEquals("Should only contain 12 methods", 12, methods.length);
547                         for(Method method: methods) {
548                             if(method.toString().contains("getIdNumber")) {
549                                 assertEquals("Didn't recieve a string", "802-14-1234", (
                                    String) method.invoke(absCIIC1));
550                             }
551                             if(method.toString().contains("getIncomeTaxRate")) {
552                                 assertTrue("Didn't recieve a double", method.invoke(absCIIC1
                                    ) instanceof Double);
553                             }
554                             if(method.toString().contains("getFirstName")) {
555                                 assertEquals("Didn't recieve a string", "Maria", (String)
                                    method.invoke(absINSO1));
556                             }
557                             if(method.toString().contains("getLastName")) {
558                                 assertEquals("Didn't recieve a string", "Santiago", (String)
                                     method.invoke(absINSO1));
559                             }
```

```
560                         if(method.toString().contains("getGender")) {
561                             assertEquals("Didn't recieve a Gender", StudentWrapper.
                                Gender.MALE, (StudentWrapper.Gender) method.invoke(absCIIC1
                                ));
562                         }
563                         if(method.toString().contains("getGPA")) {
564                             assertEquals("Didn't recieve a Number", 3.67, (Double)
                                method.invoke(absCIIC1), 1E-10);
565                         }
566                         if(method.toString().contains("setIdNumber")) {
567                             method.invoke(absCIIC1, "1111");
568                             assertEquals("The id wasn't updated", "1111", ciic1.
                                getIdNumber());
569                             method.invoke(absCIIC1, "802-14-1234");
570                         }
571                         if(method.toString().contains("setFirstName")) {
572                             method.invoke(absINSO1, "Al");
573                             assertEquals("The name wasn't updated", "Al", inso1.
                                getFirstName());
574                             method.invoke(absINSO1, "Maria");
575                         }
576                         if(method.toString().contains("setLastName")) {
577                             method.invoke(absCIIC1, "Jones");
578                             assertEquals("The name wasn't updated", "Jones", ciic1.
                                getLastName());
579                             method.invoke(absCIIC1, "Santiago");
580                         }
581                         if(method.toString().contains("setGender")) {
582                             method.invoke(absINSO1, StudentWrapper.Gender.MALE);
583                             assertEquals("The gender wasn't updated", StudentWrapper.
                                Gender.MALE, inso1.getGender());
584                             method.invoke(absINSO1, StudentWrapper.Gender.FEMALE);
585                         }
586                         if(method.toString().contains("getGPA")) {
587                             method.invoke(absINSO1, 0);
588                             assertEquals("The gender wasn't updated", 0, inso1.getGPA(),
                                 1E-10);
589                             method.invoke(absINSO1, 3.89);
590                         }
591                         if(method.toString().contains("getSpecialtyGPA")) {
592                             if(!method.toString().contains("abstract")) {
593                                 hasMethods = false;
594                                 break;
595                             }
596                         }
597                         if(method.getName().contains("getSpecialtyGPA")) {
598                             if(!method.toString().contains("abstract")) {
599                                 hasMethods = false;
600                                 break;                              }
601                         }
602                         else {
603                             hasMethods = false;
604                             break;
605                         }
606                     }
607                 }
608             }
609 
610         } catch (Exception e) {
611             fail(e.toString());
612         }
613         assertFalse("These abstract class has missing methods", hasMethods);
614 
615     }
616 
617     @Test
```

```
618         public void testRefactoring() {
619             boolean hasMethods = false;
620             try {
621                 Class<?> c = Class.forName("StudentWrapper");
622                 Class<?> cl[] = c.getDeclaredClasses();
623                 for(Class<?> cls : cl) {
624                     if(cls.toString().contains("CIIC") || cls.toString().contains("INSO")) {
625                         Method[] methods = cls.getDeclaredMethods();
626                         for(Method method: methods) {
627                             if(method.getName().contains("getIdNumber") ||
628                                     method.getName().contains("getFirstName") ||
629                                     method.getName().contains("getLastName") ||
630                                     method.getName().contains("getGender") ||
631                                     method.getName().contains("getGPA") ||
632                                     method.getName().contains("setIdNumber") ||
633                                     method.getName().contains("setFirstName") ||
634                                     method.getName().contains("setLastName") ||
635                                     method.getName().contains("setGender")) {
636                                 hasMethods = true;
637                                 break;
638                             }
639                         }
640                     }
641                 }
642 
643             } catch (Exception e) {
644                 fail(e.toString());
645             }
646             assertFalse("The subclases still contain methods that should be in the abstract
                class", hasMethods);
647         }
648 
649         @Test
650         public void testGetSpecialtyGPA() {
651             assertEquals("Wrong specialty GPA for CIIC student", ciic1.getCIICGPA(), ciic1.
                getSpecialtyGPA(),1e-5);
652             assertEquals("Wrong specialty GPA for INSO student", inso1.getINSOGPA(), inso1.
                getSpecialtyGPA(),1e-5);
653         }
654 
655         @Test
656         public void testCloneableInterface() {
657 
658             boolean interfaceFound = false;
659             try {
660                 Class<?> cls = StudentWrapper.AbstractStudent.class;
661                 Class<?>[] inters = cls.getInterfaces();
662                 for(Class<?> inter: inters) {
663                     if(inter.getName().contains("Cloneable")) {
664                         interfaceFound = true;
665                     }
666                 }
667             } catch (Exception e) {
668                 fail(e.toString());
669             }
670             assertTrue("Cloneable interface not implemented by abstract class",
                interfaceFound);
671 
672             StudentWrapper.AbstractStudent ciic1Copy = (StudentWrapper.AbstractStudent)
                ciic1.clone();
673             StudentWrapper.AbstractStudent ciic2Copy = (StudentWrapper.AbstractStudent)
                ciic2.clone();
674 
675             assertTrue("The copy generated must be an instance of the CIICStudent Class",
                ciic1Copy instanceof StudentWrapper.CIICStudent);
676             assertTrue("The copy generated must be an instance of the CIICStudent Class",
```

```
            ciic2Copy instanceof StudentWrapper.CIICStudent);
677
678         assertFalse("The copy should not share the same instance", ciic1 == ciic1Copy);
679         assertFalse("The copy should not share the same instance", ciic2 == ciic2Copy);
680
681         assertFalse("The copy should not share the same position instance", ciic1.
            getIdNumber() == ((StudentWrapper.CIICStudent)ciic1Copy).getIdNumber());
682         assertFalse("The copy should not share the same position instance", ciic2.
            getIdNumber() == ((StudentWrapper.CIICStudent)ciic2Copy).getIdNumber());
683
684         assertFalse("The copy should not share the same position instance", ciic1.
            getLastName() == ((StudentWrapper.CIICStudent)ciic1Copy).getLastName());
685         assertFalse("The copy should not share the same position instance", ciic2.
            getFirstName() == ((StudentWrapper.CIICStudent)ciic2Copy).getFirstName());
686
687         assertEquals("These copies should be identical", ciic1.getIdNumber(), ((
            StudentWrapper.CIICStudent)ciic1Copy).getIdNumber());
688         assertEquals("These copies should be identical", ciic2.getIdNumber(), ((
            StudentWrapper.CIICStudent)ciic2Copy).getIdNumber());
689     }
690
691     @Test
692     public void testDefineTaxPayerInterface() {
693
694         boolean interfaceFound = false;
695         Class<?> taxPayerInterface=null;
696         try {
697             Class<?> cls = StudentWrapper.class;
698             Class<?>[] inters = cls.getClasses();
699             for(Class<?> inter: inters) {
700                 if(inter.getName().contains("IncomeTaxPayer")) {
701                     interfaceFound = true;
702                     taxPayerInterface = inter;
703                 }
704             }
705         } catch (Exception e) {
706             fail(e.toString());
707         }
708         assertTrue("IncomeTaxPayer interface not implemented by abstract class",
            interfaceFound);
709         Method[] methods = taxPayerInterface.getMethods();
710         assertEquals("TaxPayerInterface should have only one method", 1, methods.length);
711         assertEquals(methods[0].getName(), "getIncomeTaxRate");
712
713     }
714
715     @Test
716     public void testImplementTaxPayerInterface() {
717
718         assertTrue("CIICStudent should be instance of IncomeTaxPayer", ciic1 instanceof
            StudentWrapper.IncomeTaxPayer);
719         assertTrue("INSOStudent should be instance of IncomeTaxPayer", inso1 instanceof
            StudentWrapper.IncomeTaxPayer);
720
721         assertEquals("INSOStudent should pay 5% tax rate", 0.05, inso1.getIncomeTaxRate
            (), 1e-5);
722         assertEquals("The TaxPayer interface does not contain getTaxRate", 0.05,(ciic1).
            getIncomeTaxRate(), 1e-5);
723
724     }
725
726     @Rule
727     public final ExpectedException exception = ExpectedException.none();
728
729
730     @Test
```

```
731         public void testFindINSOStudent() {
732             assertEquals("Top student of empty list should be null", false, inso1.
                findINSOStudent(emptyINSOList, 0));
733             assertTrue("Existing student not found in list", inso4.findINSOStudent(
                allINSOList, 0));
734             exception.expect(StackOverflowError.class);
735             assertEquals("Top student in list should be Juan", ciic1, inso1.findINSOStudent(
                hugeINSOList, 0));
736         }
737 
738         @Test
739         public void testFindTopCIICStudent() {
740             assertEquals("Top student of empty list should be null", null, StudentWrapper.
                CIICStudent.getTopCIICStudent(0,emptyCIICList));
741             assertEquals("Top student in list should be Juan", ciic3, StudentWrapper.
                CIICStudent.getTopCIICStudent(0,allCIICList));
742             exception.expect(StackOverflowError.class);
743             assertEquals("Top student in list should be Juan", ciic1, StudentWrapper.
                CIICStudent.getTopCIICStudent(0,hugeCIICList));
744         }
745 
746     }
747 
748     // ********************************************************************
```