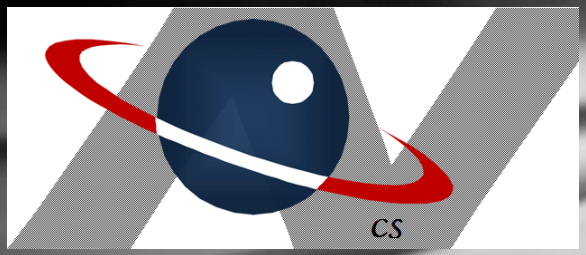
O(N) CS LESSONS

Lesson 10D – Composition



By John B. Owen

Table of Contents

Objectives

- In this lesson we will explore the concept of Composition by defining a Course class and adding that as new instance field to the Student class.

Course class

- Think about the information needed for a Course class.
- For now we'll just include two items:
  - The name of the course, like "Spanish"
  - The grade field, like 'A', which represents the student's current single letter grade in the course.

Course class

- First start with the header
- What should it be?
- Does it "extend" anything?

Course class

- First start with the header
- What should it be?
- Does it "extend" anything?
- Here is the header…

Tester.java | Person.java | **Course.java**

```
1 public class Course
2 {
3 }
```

Course class

- It does not inherit anything other than the Object class, which is automatically inherited, so the "extends" statement is not necessary

Tester.java | Person.java | **Course.java**

```
1  public class Course
2  {
3  }
```

Course class

- Testing…default output

Tester.java | Person.java | Course.java

```
1  public class Tester{
2      public static void main(String [] args) {
3          Course c = new Course();
4          System.out.println(c);
5      }
6  }
```

Tester.java | Person.java | Course.java

```
1  public class Course
2  {
3  }
```

```
Course@659e0bfd
Press any key to continue..
```

Add instance fields, override toString

- Now we define the instance fields, name of the course and current grade, and over-ride the toString method.

Tester.java | Person.java | **Course.java**

```
1  public class Course
2  {
3      private String name;
4      private char grade;
5      //override toString method
6      public String toString(){
7          return "Course:"+name+"| Grade:"+grade+"|";
8      }
9  }
```

Add instance fields, override toString

- Testing…default values, customized output

Tester.java | Person.java | Course.java

```
1  public class Tester{
2      public static void main(String [] args) {
3          Course c = new Course();
4          System.out.println(c);
5      }
6  }
```

```
Course:null| Grade: |
Press any key to continu
```

Tester.java | Person.java | Course.java

```
1  public class Course
2  {
3      private String name;
4      private char grade;
5      //override toString method
6      public String toString(){
7          return "Course:"+name+"| Grade:"+grade+"|";
8      }
9  }
```

Add instance fields, override toString

- The default value for a String is null and for a char an empty space.

```
Course:null| Grade: |
Press any key to continu
```

Tester.java | Person.java | Course.java

```
1  public class Course
2  {
3      private String name;
4      private char grade;
5      //override toString method
6      public String toString(){
7          return "Course:"+name+"| Grade:"+grade+"|";
8      }
9  }
```

Add constructors

- Next we need constructors, default and overloaded versions.

```
5       //default constructor
6       public Course(){
7           name = "Spanish";
8           grade = 'A';
9       }
10      //two-parameter constructor
11      public Course(String n, char g){
12          name = n;
13          grade = g;
14      }
```

Add constructors

- Testing…both constructors

Tester.java | Person.java | Course.java

```
1 public class Tester{
2     public static void main(String [] args)
3         Course c = new Course();
4         System.out.println(c);
5         c = new Course("Math",'B');
6         System.out.println(c);
7     }
8 }
```

```
Course:Spanish| Grade:A|
Course:Math| Grade:B|
Press any key to continue...
```

Add access methods

- Now the access methods…

Tester.java | Person.java | **Course.java**

```
 6      public String getName(){
 7          return name;
 8      }
 9      public char getGrade(){
10          return grade;
11      }
12      public void setName(String n){
13          name = n;
14      }
15      public void setGrade(char g){
16          grade = g;
17      }
```

Add access methods

- Testing accessor methods…

Tester.java | Person.java | Course.java

```
1  public class Tester{
2      public static void main(String [] args) {
3          Course c = new Course();
4          System.out.println(c.getName());
5          System.out.println(c.getGrade());
6          c = new Course("Math",'B');
7          System.out.println(c.getName());
8          System.out.println(c.getGrade());
9      }
10 }
```

```
Spanish
A
Math
B
Press any key to continu
```

Add access methods

- Testing modifier methods…

Tester.java | Person.java | Course.java | Student.java

```
1  public class Tester{
2      public static void main(String [] args) {
3          Course c = new Course();
4          c.setName("Physics");
5          c.setGrade('F');
6          System.out.println(c);
7      }
8  }
```

```
Course:Physics| Grade:F|
Press any key to continue...
```

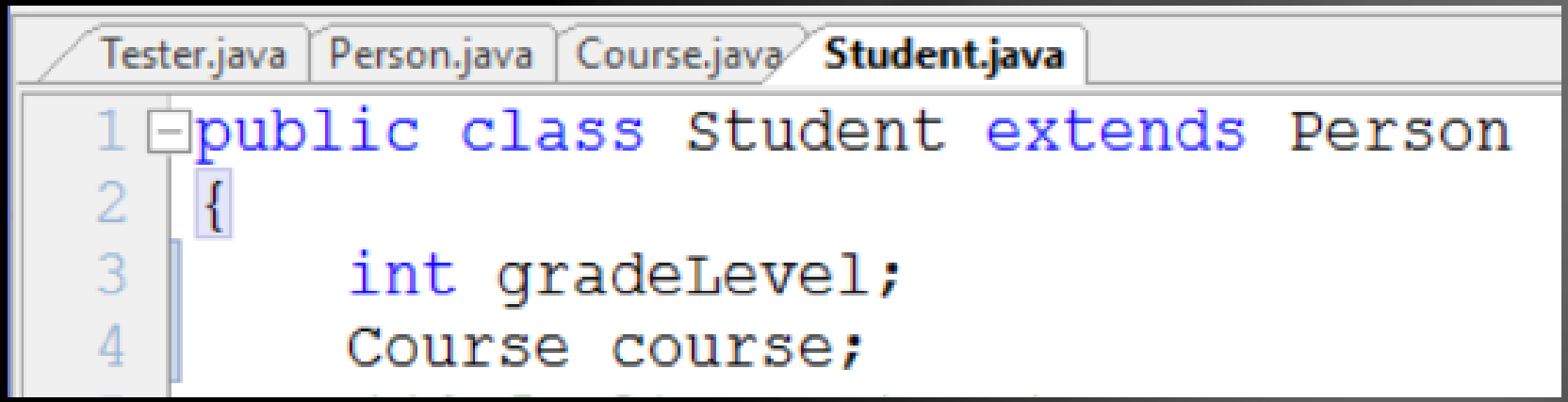
Completed Course class definition

Tester.java | Person.java | **Course.java**

```
1  public class Course{
2      private String name;
3      private char grade;
4      //access methods
5      public String getName(){
6          return name;
7      }
8      public char getGrade(){
9          return grade;
10     }
11     public void setName(String n){
12         name = n;
13     }
14     public void setGrade(char g){
15         grade = g;
16     }
17     //constructors
18     public Course(){
19         name = "Spanish";
20         grade = 'A';
21     }
22     public Course(String n, char g){
23         name = n;
24         grade = g;
25     }
26     //toString override
27     public String toString(){
28         return "Course:"+name+"| Grade:"+
29     }
30 }
```

Add Course field to Student class

- Now we add a Course field to the Student class, an example of Composition.
- The Student class now "has a" Course.



Add Course field to Student class

- Inheritance is still involved because the Student class still "is a" Person.

Tester.java | Person.java | Course.java | **Student.java**

```
1  public class Student extends Person
2  {
3      int gradeLevel;
4      Course course;
```

Adjust toString method

- We tack on the Course to the end of the toString return statement…

Tester.java | Person.java | Course.java | **Student.java**

```
23      //override toString method
24      public String toString(){
25          return "Student "+getName()+" is "
26                  +getAge()+" years old "
27                  +"\nand is in grade "+gradeLevel+".\n"
28                   +course;
29      }
30  }
```

Adjust toString method

- …and then test that output.

Tester.java | Person.java | Course.java | **Student.java**

```
23      //override toString method
24      public String toString(){
25          return "Student "+getName()+" is "
26                 +getAge()+" years old "
27                 +"\nand is in grade "+gradeLevel+".\n"
28                  +course;
29      }
30  }
```

```
Student Joey is 18 years old
and is in grade 12.
null
Press any key to continue...
```

Adjust toString method

- The output for the Course data field shows null since it is an object.

Tester.java | Person.java | Course.java | **Student.java**

```
23      //override toString method
24      public String toString(){
25          return "Student "+getName()+" is "
26                 +getAge()+" years old "
27                 +"\nand is in grade "+gradeLevel+".\n"
28                  +course;
29      }
30  }
```

```
Student Joey is 18 years old
and is in grade 12.
null
Press any key to continue...
```

Adjust toString method

- …which has not been instantiated by the Student default constructor.

Tester.java | Person.java | Course.java | **Student.java**

```
5       //default constructor
6       public Student(){
7           super();
8           gradeLevel = 12;
9       }
```

```
Student Joey is 18 years old
and is in grade 12.
null
Press any key to continue...
```

Adjust Student default constructor

- We instantiate a default Course inside the Student constructor…

Tester.java | Person.java | Course.java | **Student.java**

```
5       //default constructor
6       public Student(){
7           super();
8           gradeLevel = 12;
9           course = new Course();
10      }
```

Adjust Student default constructor

- We instantiate a default Course inside the Student constructor…

Tester.java | Person.java | Course.java | **Student.java**

```
5       //default constructor
6       public Student(){
7           super();
8           gradeLevel = 12;
9           course = new Course();
10      }
```

- …which results in this output

```
Student Joey is 18 years old
and is in grade 12.
Course:Spanish| Grade:A|
Press any key to continue...
```

Overloaded Student constructor

- Here is the overloaded constructor…

Tester.java | Person.java | Course.java | **Student.java**

```
11      //four-parameter constructor
12      public Student(String s, int a, int g, Course c){
13          super(s,a);
14          gradeLevel = g;
15          course = c;
16      }
```

Overloaded Student constructor

- …an adjusted Tester class

Tester.java | Person.java | Course.java | Student.java

```
1 public class Tester{
2     public static void main(String [] args) {
3         Course c = new Course("Algebra",'C');
4         Student me = new Student("Jill",14,8,c);
5         System.out.println(me);
6     }
7 }
```

- …and output result

```
Student Jill is 14 years old
and is in grade 8.
Course:Algebra| Grade:C|
Press any key to continue...
```

Lesson Summary

- To summarize, a Course class was defined, with two fields, name and grade.
- A toString method was overridden to control the output.
- Constructors and access methods were defined.

Lesson Summary

- A Course field was added to the Student class
- Adjustments to the Student class methods (toString, constructors, access methods) were made to accommodate the new field.
- Testing was performed all along the way.

Lesson Summary

- Over the last several lessons, you have seen the development process of a simple Student class, showing aspects of
  - Inheritance and Composition,
  - Polymorphism (overriding and overloading)
  - Encapsulation and Information Hiding

Lesson Summary

- These are the foundations of OOP, and if you can understand these fundamental processes, you will be well prepared to continue further study into this important and exciting field of computer science and programming!

Lab

- Again using the example in this lesson, define an Organization class to be added as a field to the Student class.
- Define a name field inside the Organization class, and another field of your own imagination, something appropriate to this class.

Lab

- An Organization might be "Band", or "ROTC", or "Student Council".
- Think about another field that would be common to all of these three examples and include that as a field in the Organization class.

Lab

- Finally, add the Organization class to the Student class as another instance field, and make adjustments to the Student class as necessary.

CONGRATULATIONS!

- Now you know how to develop an OOP class involving aspects of Inheritance and Composition, Polymorphism, and Encapsulation and Information Hiding, using a step by step development process to design a fairly complex class, using testing along the way.

Thanks, and have fun!

To order supplementary materials for all the lessons in this series, including lesson examples, lab solutions, quizzes, tests, and unit reviews, visit the O(N)CS Lessons website, or contact me at

John B. Owen
captainjbo@gmail.com