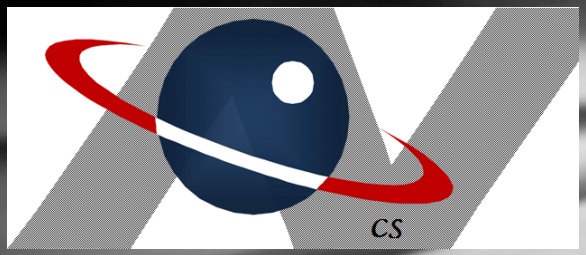
O(N) CS LESSONS

Lesson 10C – Inheritance



By John B. Owen

Table of Contents

Objectives

- In this lesson we will explore further the concept of Inheritance by renaming the previously defined Student class as the Person class, which will then serve as a super class for a new version of the Student class, which will inherit the Person class.

Person class

- First we rename the Student class to be a Person class, with all the same characteristics, i.e., name, age, etc.
- See the next two slides for that class definition.

Person class

Person.java | Tester.java

```
1  public class Person
2  {
3      private String name;
4      private int age;
5      //Modifiers/mutators
6      public void setName(String n){
7          name = n;
8      }
9      public void setAge(int a){
10         age = a;
11     }
12     //Accessor methods
13     public String getName(){
14         return name;
15     }
16     public int getAge(){
17         return age;
18     }
```

Person class

```
17          return age;
18      }
19      //override toString method
20      public String toString()     {
21          return "Person: "+name+" Age: "+age;
22      }
23      //default constructor
24      public Person() {
25          name = "Joey";
26          age = 18;
27      }
28      //two-parameter constructor
29      public Person(String s, int a)   {
30          name = s;
31          age = a;
32      }
33  }
```

Person class

- As you can see, it is identical to the previously defined Student class, complete with
  - private name and age fields
  - two constructors (named Person)
  - overridden toString method (slightly modified)
  - accessors and modifiers

Tester class

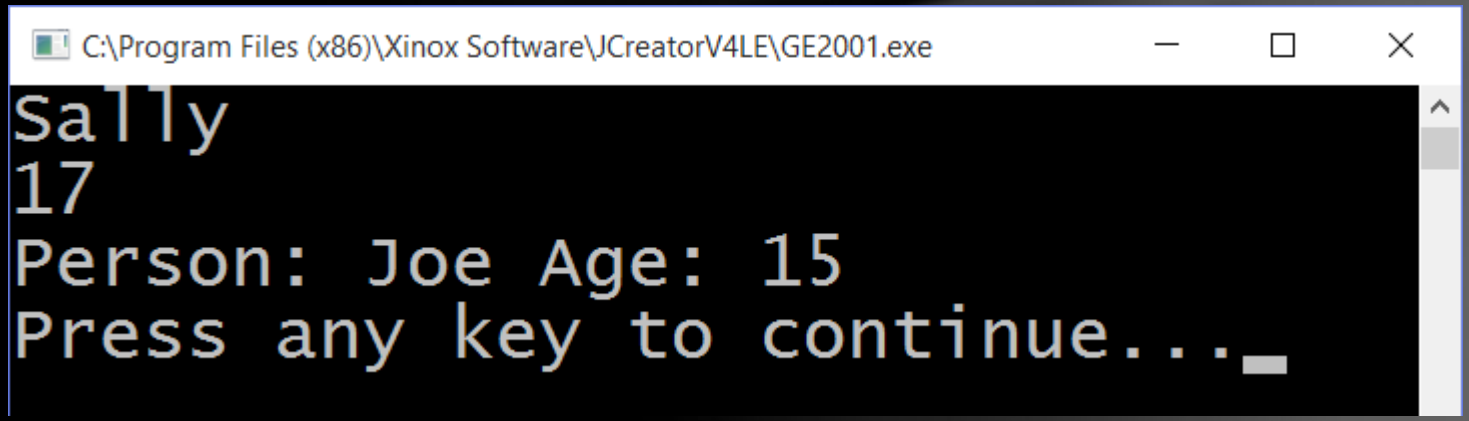
- The Tester class is now in its own file, separate from the Person class.

Person.java | Tester.java

```
1  public class Tester{
2      public static void main(String [] args) {
3          Person me = new Person("Sally",17);
4          System.out.println(me.getName());
5          System.out.println(me.getAge());
6          me.setName("Joe");
7          me.setAge(15);
8          System.out.println(me);
9      }
10 }
```

Altered toString method

- Note in the output the one minor adjustment to the toString method, saying the word "Person" instead of "Name".



New Student class

- Now we define a new version of the Student class, inheriting the Person class, and then adding a new field particular to a Student, namely their grade level in school (K-12).

New Student class

- At first we simply define the Student class with the phrase "extends Person" to indicate inheritance of the Person class.

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

```
1  public class Student extends Person
2  {
3  }
```

New Student class

- This means the Student now "owns" everything the Person class owns, including the Object class methods the Person class inherited.

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

```
1 public class Student extends Person
2 {
3 }
```

New Student class

- From the Tester class, a new Student is constructed and output.

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

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

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

```
1 public class Student extends Person
2 {
3 }
```

C:\Program Files (x86)\Xinox Software\JCreatorV4LE\GE2001.exe

```
Person: Joey Age: 18
Press any key to continue...
```

New Student class

- Note carefully that the Student class contains NOTHING except for the header.

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

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

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

```
1  public class Student extends Person
2  {
3  }
```

C:\Program Files (x86)\Xinox Software\JCreatorV4LE\GE2001.exe

```
Person: Joey Age: 18
Press any key to continue...
```

New Student class

- How is this output achieved when NOTHING is in the Student class???

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

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

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

```
1  public class Student extends Person
2  {
3  }
```

C:\Program Files (x86)\Xinox Software\JCreatorV4LE\GE2001.exe

```
Person: Joey Age: 18
Press any key to continue...
```

New Student class

- The answer is simple…the methods of the Person class are doing all the work!!!

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

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

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

```
1  public class Student extends Person
2  {
3  }
```

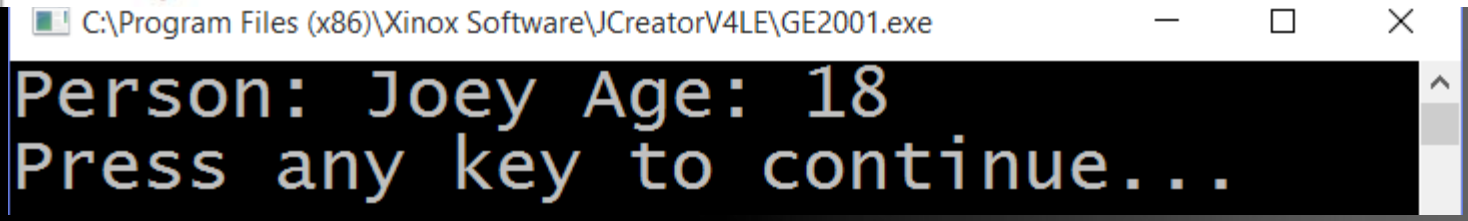
C:\Program Files (x86)\Xinox Software\JCreatorV4LE\GE2001.exe

```
Person: Joey Age: 18
Press any key to continue...
```

New Student class

- The compiler uses the default constructor of the Person class since it found nothing in the Student class, but noted that the Person class was extended.

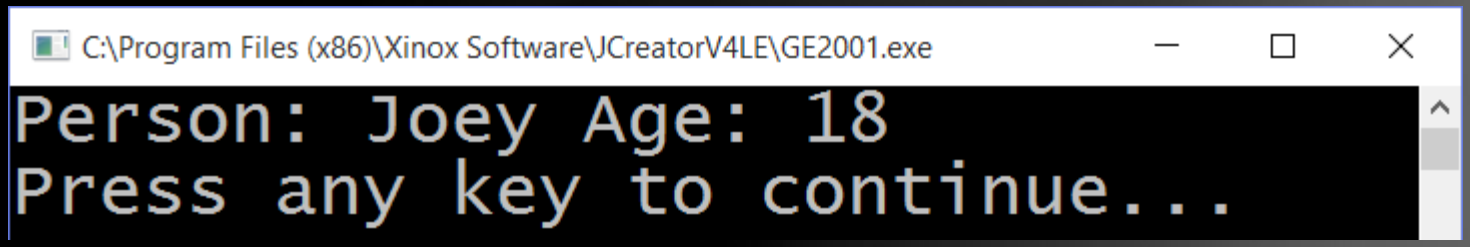
```
23      //default constructor
24      public Person() {
25          name = "Joey";
26          age = 18;
27      }
```



New Student class

- The compiler also used the toString method from the Person class since the Student class had not overridden it.

```
18      }
19      //override toString method
20      public String toString()     {
21          return "Person: "+name+" Age: "+age;
22      }
```



super.toString()

- Let's override the toString method in the Student class…study it carefully.

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

```
1  public class Student extends Person
2  {
3      //override toString method
4      public String toString()     {
5          return "Student-"+super.toString();
6      }
7  }
```

```
C:\Program Files (x86)\Xinox Software\JCreatorV4LE\GE2001.exe
Student-Person: Joey Age: 18
Press any key to continue...
```

super.toString()

- What's happening here???

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

```
1  public class Student extends Person
2  {
3      //override toString method
4      public String toString()     {
5          return "Student-"+super.toString();
6      }
7  }
```

C:\Program Files (x86)\Xinox Software\JCreatorV4LE\GE2001.exe

```
Student-Person: Joey Age: 18
Press any key to continue...
```

super.toString()

- The word "Student-" is combined with the Person version of the toString method!!

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

```
1  public class Student extends Person
2  {
3      //override toString method
4      public String toString()     {
5          return "Student-"+super.toString();
6      }
7  }
```

```
C:\Program Files (x86)\Xinox Software\JCreatorV4LE\GE2001.exe
Student-Person: Joey Age: 18
Press any key to continue...
```

super.toString()

- The super.toString() part refers directly to the toString method of the Person class.

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

```
1  public class Student extends Person
2  {
3      //override toString method
4      public String toString()     {
5          return "Student-"+super.toString();
6      }
7  }
```

```
C:\Program Files (x86)\Xinox Software\JCreatorV4LE\GE2001.exe
Student-Person: Joey Age: 18
Press any key to continue...
```

super.toString()

- Using the word super refers to the parent class.

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

```
1  public class Student extends Person
2  {
3      //override toString method
4      public String toString()     {
5          return "Student-"+super.toString();
6      }
7  }
```

```
C:\Program Files (x86)\Xinox Software\JCreatorV4LE\GE2001.exe
Student-Person: Joey Age: 18
Press any key to continue...
```

Super class data access

- Let's try a different toString version.
- Will this work??

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

```
1  public class Student extends Person
2  {
3      //override toString method
4      public String toString()     {
5  //         return "Student-"+super.toString();
6          return "Student "+name+" is "
7                  +age+" years old.";
8      }
```

Super class data access

- No, this will not work!!
- Why not??
- Doesn't the Student class "own" the name and age fields??
- Why are there access errors??

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

Task View

Build Report

Message	Folder	Location
Resource: Student.java		
error: name has private access in Person	C:\Users\John\Desktop_Tea...	line 6
error: age has private access in Person	C:\Users\John\Desktop_Tea...	line 7

Super class data access

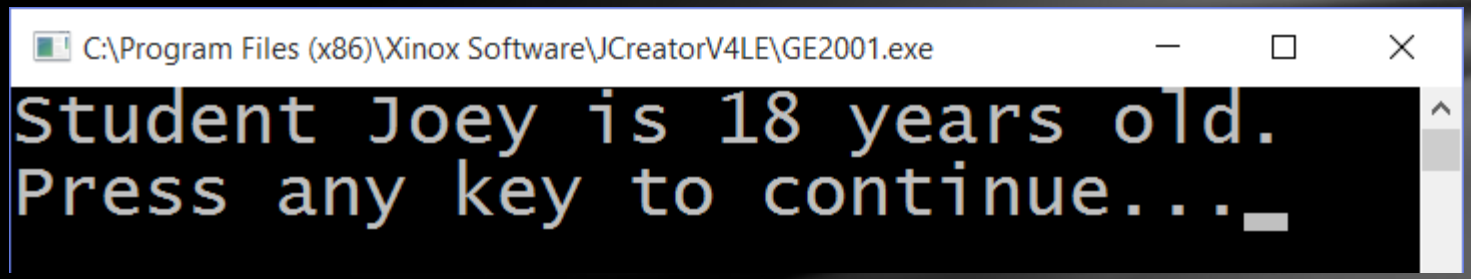
- Here's why this will not work.
- Although the Student class does indeed "own" the Parent class fields, since they are private, they must be accessed using the special access methods.

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

```
1  public class Student extends Person
2  {
3      //override toString method
4      public String toString()     {
5  //         return "Student-"+super.toString();
6          return "Student "+getName()+" is "
7                  +getAge()+" years old.";
8      }
```

Super class data access

- Now it works, since the getName and getAge accessor methods were used.



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

```
1  public class Student extends Person
2  {
3      //override toString method
4      public String toString()    {
5  //      return "Student-"+super.toString();
6          return "Student "+getName()+" is "
7                  +getAge()+" years old.";
8      }
```

Sub class data field

- Let's add a new field to the Student class and adjust the toString method.

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

```
1  public class Student extends Person
2  {
3      int gradeLevel;
4      //override toString method
5      public String toString(){
6  //      return "Student-"+super.toString();
7          return "Student "+getName()+" is "
8                  +getAge()+" years old "
9                  +"\nand is in grade "+gradeLevel+".";
10     }
11 }
```

```
Student Joey is 18 years old
and is in grade 0.
Press any key to continue...
```

Sub class data field

- This works OK, but the output does not make sense. Why??

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

```
1  public class Student extends Person
2  {
3      int gradeLevel;
4      //override toString method
5      public String toString(){
6  //      return "Student-"+super.toString();
7          return "Student "+getName()+" is "
8                  +getAge()+" years old "
9                  +"\nand is in grade "+gradeLevel+".";
10     }
11 }
```

```
Student Joey is 18 years old
and is in grade 0.
Press any key to continue...
```

Sub class data field

- First of all, since gradeLevel is a direct member of Student, an accessor method is not required.

```
3       int gradeLevel;
4       //override toString method
5       public String toString(){
6   //      return "Student-"+super.toString();
7           return "Student "+getName()+" is "
8                   +getAge()+" years old "
9                   +"\nand is in grade "+gradeLevel+".";
10      }
11  }
```

```
Student Joey is 18 years old
and is in grade 0.
Press any key to continue...
```

Sub class data field

- Secondly, the value of gradeLevel is not appropriate for a default value, given the fact that Joey is 18 years old.

```
3       int gradeLevel;
4       //override toString method
5       public String toString(){
6   //      return "Student-"+super.toString();
7           return "Student "+getName()+" is "
8                   +getAge()+" years old "
9                   +"\nand is in grade "+gradeLevel+".";
10      }
11  }
```

```
Student Joey is 18 years old
and is in grade 0.
Press any key to continue...
```

Sub class default constructor

- To solve this we revise the default constructor, not in the Person class, but rather in the Student class, as shown below.

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

```
1  public class Student extends Person
2  {
3      int gradeLevel;
4      //default constructor
5      public Student(){
6          super();
7          gradeLevel = 12;
8      }
```

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

Sub class default constructor

- This is another example of over-riding, a feature of polymorphism.

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

```
1  public class Student extends Person
2  {
3      int gradeLevel;
4      //default constructor
5      public Student(){
6          super();
7          gradeLevel = 12;
8      }
```

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

Sub class default constructor

- When a class inherits a super class, the first statement in the default constructor MUST be super(), which in this case means, *"first construct a Person object, then do more related to the Student object"*.

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

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

Sub class default constructor

- Now the output makes more sense, given the fact that Joey, being 18 years old, is in grade 12.

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

```
1  public class Student extends Person
2  {
3      int gradeLevel;
4      //default constructor
5      public Student(){
6          super();
7          gradeLevel = 12;
8      }
```

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

Sub class overloaded constructor

- What about overriding the two-parameter constructor?
- Let's do it!

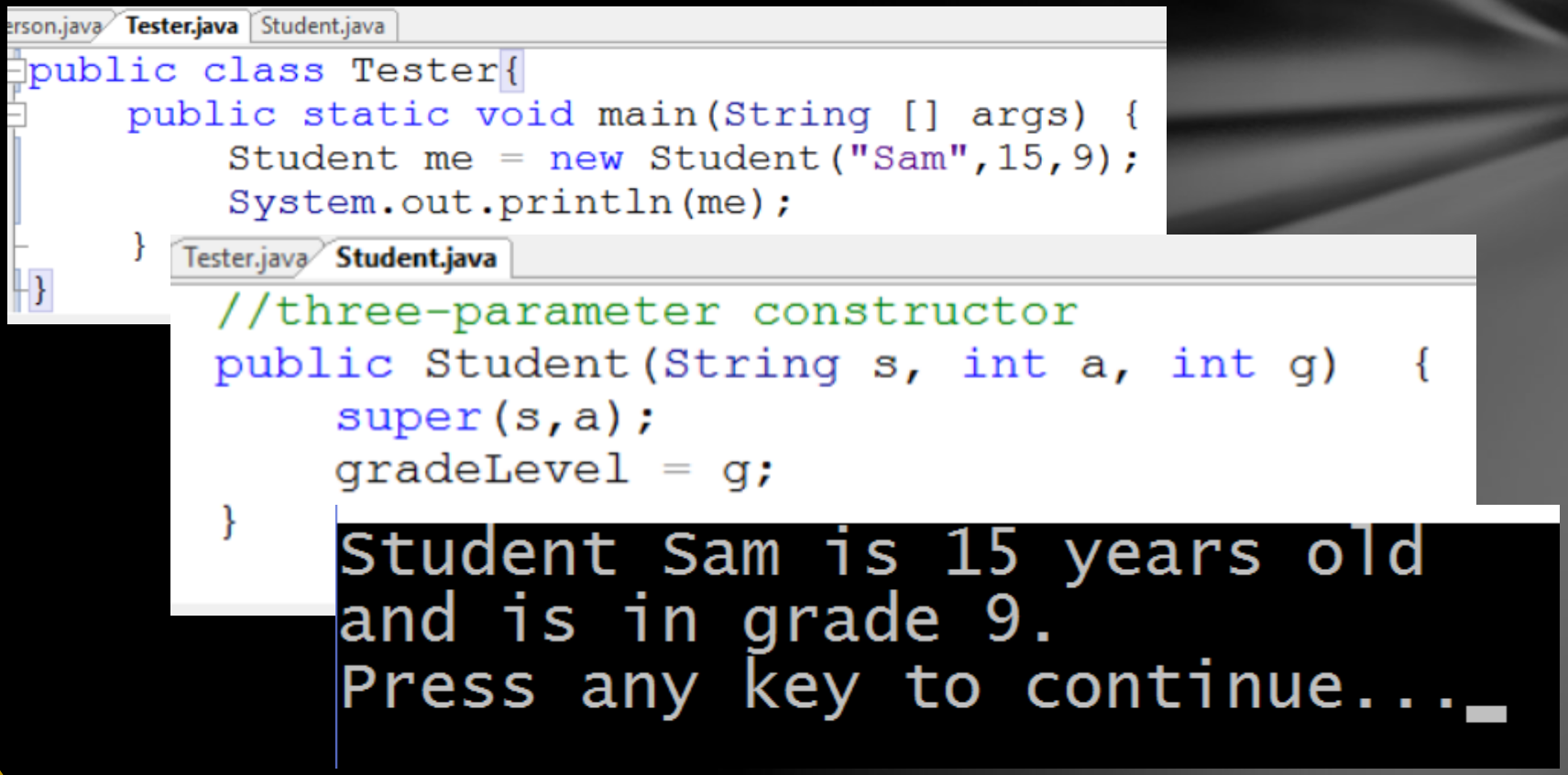
Person.java | Tester.java | Student.java

```
28      //two-parameter constructor
29      public Person(String s, int a)   {
30          name = s;
31          age = a;
32      }
33  }
```

- Above is the Person version, which requires two parameters.
- What does the Student version look like?

Sub class overloaded constructor

- It looks like this.



Sub class overloaded constructor

- Since the Student class actually owns three instance fields, all three need to be assigned default values.

Tester.java | **Student.java**

```
//three-parameter constructor
public Student(String s, int a, int g)   {
    super(s,a);
    gradeLevel = g;
}
```

```
Student Sam is 15 years old
and is in grade 9.
Press any key to continue...
```

Sub class overloaded constructor

- The first statement is a call to the Person two-parameter constructor, and then the gradeLevel field is assigned.

Tester.java | **Student.java**

```
//three-parameter constructor
public Student(String s, int a, int g)   {
    super(s,a);
    gradeLevel = g;
}
```

```
Student Sam is 15 years old
and is in grade 9.
Press any key to continue...
```

Accessor for sub class data field

- The final step in this class definition is to define the accessor and modifier method for the Student gradeLevel field.

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

```
14      public void setGradeLevel(int g){
15          gradeLevel = g;
16      }
17      public int getGradeLevel(){
18          return gradeLevel;
19      }
```

Parent class variable issues

- Now, let's do some testing.
- This works just fine.

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

```
1  public class Tester{
2      public static void main(String [] args) {
3          Student me = new Student("Sam",15,9);
4          System.out.println(me);
5          System.out.println(me.getGradeLevel());
6      }
7  }
```

```
Student Sam is 15 years old
and is in grade 9.
9
Press any key to continue...
```

Parent class variable issues

- This doesn't. Why not??
- Look carefully. Something has changed.

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

```
1  public class Tester{
2      public static void main(String [] args) {
3          Person me = new Student("Sam",15,9);
4          System.out.println(me);
5          System.out.println(me.getGradeLevel());
```

Build Output

```
--------------------Configuration: <Default>--------------------
C:\Users\John\Desktop\_TeachCS\MOOC\MOOCDevelopment\Tester.java:5: error: cannot find symbo
        System.out.println(me.getGradeLevel());
                           ^
  symbol:   method getGradeLevel()
  location: variable me of type Person
1 error

Process completed.
```

Parent class variable issues

- Did you spot it?
- The object variable is now a Person, even though the object is still a Student.

```
3       Person me = new Student("Sam",15,9);
4       System.out.println(me);
5       System.out.println(me.getGradeLevel());
```

```
--------------------Configuration: <Default>--------------------
C:\Users\John\Desktop\_TeachCS\MOOC\MOOCDevelopment\Tester.java:5: error: cannot find symbo
        System.out.println(me.getGradeLevel());
                           ^
  symbol:   method getGradeLevel()
  location: variable me of type Person
1 error

Process completed.
```

Parent class variable issues

- Why is that a problem?
- Because gradeLevel is not a method directly known to the Person class.
- It is strictly a Student class method.

```
5           System.out.println(me.getGradeLevel());
```

Build Output

```
--------------------Configuration: <Default>--------------------
C:\Users\John\Desktop\_TeachCS\MOOC\MOOCDevelopment\Tester.java:5: error: cannot find symbo
        System.out.println(me.getGradeLevel());
                           ^
  symbol:   method getGradeLevel()
  location: variable me of type Person
1 error

Process completed.
```

Parent class variable issues

```
Person me = new Student("Sam",15,9);
```

- Generally speaking, when you declare a super class variable (Person, in this case), and refer to a sub class object (Student, in this case), you can only access methods and data provided by the super class through that variable.

Parent class variable issues

- Anything the child class develops in addition to the inherited features are not known by or available to the parent class.
- In this case, the Person class does not know about the gradeLevel data field or accessor methods, causing an access error.

Parent class variable issues

- To access gradeLevel, you must use a Student variable to reference the Student object.

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

```
1 public class Tester{
2     public static void main(String [] args) {
3         Student me = new Student("Sam",15,9);
4         System.out.println(me);
5         System.out.println(me.getGradeLevel());
6     }
7 }
```

Parent class variable issues

- A parent class variable can refer to a child class, but only has direct knowledge of the features it has provided, including any methods that have been over-ridden.
- In this case, the name and age fields and the toString method are the only elements the Person class variable can access.

Parent class variable issues

- However, for any over-ridden methods, since the parent knows those methods exist, the compiler will use the **sub-class version** and not the original version provided by the parent.
- In this case, the more specific version of the toString method is used, even if over-ridden in the Student class.

Lesson Summary

- In this lesson, you learned more about Inheritance, dealing specifically with issues of access between the super and sub classes related to inherited data and access methods, as well as new sub class data and access methods.

Lab

- Expanding on the example provided in this lesson, add another field to the Student class called honors, indicating whether or not the student is an honors level student (this will be of type boolean, whose default value is false).

Lab

- Make all necessary adjustments and test the program accordingly, following the same process as was used when adding the gradeLevel field throughout this lesson.

Lab

- Specific tasks:
  - Define the honors field
  - Define access methods for the honors field
  - Adjust the toString method
  - Adjust all constructors (default and overloaded)
  - Test all new aspects along the way

CONGRATULATIONS!

- Now you know how to develop a sub class with additional features beyond those inherited from the super class.
- You are also aware of access issues involving the super and sub classes.

CONGRATULATIONS!

- In the next lesson we will expand the Student class to include a Course object as a data item, introducing the idea of Composition.

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