

## Lab Assignment 06, Object-Oriented Programming, CSE 271, Spring 2020

Department of Computer Science and Engineering, Miami University

### Inheritance (Extends, Super, Overloading, Overriding) and JUnit

In this lab, you will practice how to implement multiple classes using inheritance hierarchy. You will also implement JUnit tester classes to test and write Javadoc comments for the classes and methods. You must use the *instanceof* operator in equals methods to make sure there won't be any **ClassCastException** when you cast objects. Please review the slides 68-79 (Lecture\_Inheritance, posted on Canvas) to write equals method.

#### Problem 1:

In this inheritance hierarchy, we have three classes, Person, Student, and Instructor. Student and Instructor classes are the subclasses of the superclass Person. Create a project in Eclipse named **Lab\_06**.

- **Class Person:**

Create a class named **Person** which has the following information (use appropriate type or class for each instance variable):

- Name, the name of the person.
- Year of birth, the year of birth of the person.

The class should also have the following methods:

- **Constructors:** Two constructors, one default constructor, which initializes the instances variables to the default values (strings to null, numbers to zero) and the second, parameterized constructor, which initializes the instances variables using the received parameters.

- **Accessor and mutator methods for all instance variables.**

- **public boolean equals(Object obj)**

The method returns true if the two persons have the same name and year of birth. False, otherwise.

**Important Note: This is an overridden equals() method.**

- **public String toString()**

The method returns a String representing the person object that includes name and year of birth. It returns a String like "I am a person whose name is **James Bond** and was born in **1997**".

- **Class Student:**

Create a class named **Student** which is a subclass of **Person** and has the following information (use appropriate type or class for each instance variable):

- Major, the major of the student.

The class should also have the following methods:

- **Constructors:** Two constructors, one default constructor, which initializes the instances variables to the default values (strings to null, numbers to zero) and the second, parameterized constructor, which initializes the instances variables using the received parameters.

**Important Note: Utilize the super() constructor appropriately. The student constructor needs to call the super constructor with the appropriate data to initialize the person's instance variables.**

- **Accessor and mutator methods for all instance variables.**

- **public boolean equals(Object obj)**

The method returns true if the two persons have the same name, year of birth, and major. False, otherwise.

**Important Note: This is an overridden equals() method. Use equals() method from the super class whenever applicable.**

- **public String toString()**

## Lab Assignment 06, Object-Oriented Programming, CSE 271, Spring 2020

Department of Computer Science and Engineering, Miami University

### Inheritance (Extends, Super, Overloading, Overriding) and JUnit

The method returns a String representing the student object that includes name, year of birth, and major. It returns a String like "I am a student majoring in **Computer Science**. I am a person whose name is **James Bond** and was born in **1997**".

**Important Note:** This is an overridden **toString()** method. Use **toString()** method from super class to avoid writing same code again. Call **super.toString()** to get a string from the super class.

- **Class Instructor:**

Create a class named **Instructor** which is a subclass of **Person** and has the following information (use appropriate type or class for each instance variable):

- Rank, the rank of the instructor (lecturer, assistant professor, associate professor, professor).

The class should also have the following methods:

- **Constructors:** Two constructors, one default constructor, which initializes the instances variables to the default values (strings to null, numbers to zero) and the second, parameterized constructor, which initializes the instances variables using the received parameters.

**Important Note:** Utilize the **super()** constructor appropriately. The instructor constructor needs to call the super constructor with the appropriate data to initialize the person's instance variables.

- **Accessor and mutator methods for all instance variables.**

- **public boolean equals(Object obj)**

The method returns true if the two instructors have the same name, year of birth, and rank. False, otherwise.

**Important Note:** This is an overridden **equals()** method. Use **equals()** method from the super class whenever applicable.

- **public String toString()**

The method returns a String representing the instructor object that includes name, year of birth, and rank. It returns a String like "I am a **lecturer**. I am a person whose name is **James Bond** and was born in **1997**".

**Important Note:** This is an overridden **toString()** method. Use **toString()** method from super class to avoid writing same code again. Call **super.toString()** to get a string from the super class.

#### Problem 2:

In this inheritance hierarchy, we have four classes, Person, Employee, Manager, and Executive.

Employee is a subclass of Person, Manager is a subclass of Employee, and Executive is a subclass of Manager.

- **Class Employee:**

Create a class named **Employee** which is a subclass of **Person** and has the following information (use appropriate type or class for each instance variable):

- Salary, the salary of the employee (for example, \$50,000.00).

The class should also have the following methods:

- **Constructors:** Two constructors, one default constructor, which initializes the instances variables to the default values (strings to null, numbers to zero) and the second, parameterized constructor, which initializes the instances variables using the received parameters.

**Important Note:** Utilize the **super()** constructor appropriately. The student constructor needs to call the super constructor with the appropriate data to initialize the person's instance variables.

## Lab Assignment 06, Object-Oriented Programming, CSE 271, Spring 2020

Department of Computer Science and Engineering, Miami University

### Inheritance (Extends, Super, Overloading, Overriding) and JUnit

- **Accessor and mutator methods for all instance variables.**
- **public boolean equals(Object obj)**  
The method returns true if the two persons have the same name, year of birth and salary. False, otherwise.  
**Important Note: This is an overridden equals() method. Use equals() method from the super class whenever applicable.**
- **public String toString()**  
The method returns a String representing the person object that includes name, year of birth, and salary. It returns a String like "I am an employee and have a salary of **\$50,000.00**. I am a person whose name is **James Bond** and was born in **1997**".  
**Important Note: This is an overridden toString() method. Use toString() method from super class to avoid writing same code again. Call super.toString() to get a string from the super class.**
- **Class Manager:**  
Create a class named **Manager** which is a subclass of **Employee** and has the following information use appropriate type or class for each instance variable):
  - Department, the department of the manager.The class should also have the following methods:
  - **Constructors:** Two constructors, one default constructor, which initializes the instances variables to the default values (strings to null, numbers to zero) and the second, parameterized constructor, which initializes the instances variables using the received parameters.  
**Important Note: Utilize the super() constructor appropriately. The student constructor needs to call the super constructor with the appropriate data to initialize the person's instance variables.**
  - **Accessor and mutator methods for all instance variables.**
  - **public boolean equals(Object obj)**  
The method returns true if the two managers have the same name, year of birth, and department. False, otherwise.  
**Important Note: This is an overridden equals() method. Use equals() method from the super class whenever applicable.**
  - **public String toString()**  
The method returns a String representing the manager object that includes name, year of birth, and department. It returns a String like "I am a manager of the **Human Resource** department. I am an employee and have a salary of **\$50,000.00**. I am a person whose name is **James Bond** and was born in **1997**".  
**Important Note: This is an overridden toString() method. Use toString() method from super class to avoid writing same code again. Call super.toString() to get a string from the super class.**
- **Class Executive:**  
Create a class named **Executive** which is a subclass of **Manager** and has the following information use appropriate type or class for each instance variable):
  - Office location, the office location of the executive (suite 300, room 420).The class should also have the following methods:
  - **Constructors:** Two constructors, one default constructor, which initializes the instances variables to the default values (strings to null, numbers to zero) and the second, parameterized constructor, which initializes the instances variables using the received parameters.

## Lab Assignment 06, Object-Oriented Programming, CSE 271, Spring 2020

Department of Computer Science and Engineering, Miami University

### Inheritance (Extends, Super, Overloading, Overriding) and JUnit

Important Note: Utilize the **super()** constructor appropriately. The student constructor needs to call the super constructor with the appropriate data to initialize the person's instance variables.

- **Accessor and mutator methods for all instance variables.**

- **public boolean equals(Object obj)**

The method returns true if the two executives have the same name, year of birth, and office location. False, otherwise.

Important Note: This is an overridden **equals()** method. Use **equals()** method from the super class whenever applicable.

- **public String toString()**

The method returns a String representing the instructor object that includes name, year of birth, and rank. It returns a String like "I am an executive and my office location is **room 420**. I am a manager of the **Human Resource** department. I am an employee and have a salary of **\$50,000.00**. I am a person whose name is **James Bond** and was born in **1997**".

Important Note: This is an overridden **toString()** method. Use **toString()** method from super class to avoid writing same code again. Call **super.toString()** to get a string from the super class.

#### Javadoc Style Comments:

You have to make Javadoc style comments for all methods and classes including parameter and return description. Once you write all the Javadoc style comments generate Javadoc using Eclipse's "Generate Javadoc" option from the "project" menu. Include the Javadoc folder "doc" with your code submission.

#### Tester Class:

Now write JUnit testers for each of the classes you define. In your **Tester** classes you have to test all the methods (constructor, accessor, mutator, equals, toString) of the classes you defined (Person, Student, Instructor, Employee, Manager, and Executive).

#### Important Note:

Make sure the file name is correct and code is well commented. No late submission. You will get zero for late submission.

#### Submission:

Submit java files and Javadoc folder "doc" to the appropriate submission folder on the Canvas by the due time. You can zip all the java files and "doc" folder together and submit one zip file.

**Lab Assignment 06, Object-Oriented Programming, CSE 271, Spring 2020**  
**Department of Computer Science and Engineering, Miami University**  
**Inheritance (Extends, Super, Overloading, Overriding) and JUnit**

**Grading Rubric:**

<b>Person</b>	
Declare private fields with appropriate type or class	1
Constructors (each worth 1 points)	2
Accessors and Mutators (each worth 0.5 points)	2
equals() method	3
toString() method	3
<b>Student</b>	
Declare private fields with appropriate type or class	1
Constructors (each worth 2 points)	4
Accessors and Mutators (each worth 0.5 points)	1
equals() method	3
toString() method	3
<b>Instructor</b>	
Declare private fields with appropriate type or class	1
Constructors (each worth 2 points)	4
Accessors and Mutators (each worth 0.5 points)	1
equals() method	3
toString() method	3
<b>Employee</b>	
Declare private fields with appropriate type or class	1
Constructors (each worth 2 points)	4
Accessors and Mutators (each worth 0.5 points)	1
equals() method	3
toString() method	3
<b>Manager</b>	
Declare private fields with appropriate type or class	1
Constructors (each worth 2 points)	4
Accessors and Mutators (each worth 0.5 points)	1
equals() method	3
toString() method	3
<b>Executive</b>	
Declare private fields with appropriate type or class	1
Constructors (each worth 2 points)	4
Accessors and Mutators (each worth 0.5 points)	1
equals() method	3
toString() method	3
<b>Testing</b>	
Test constructors (each worth 0.5 points)	6
Test accessors and mutators (each worth 0.5 points)	7
Test equals and toString methods (each worth 1 points)	12
<b>Javadoc</b>	
Javadoc style comments	4
Submitted correct Javadoc files [Only deduction]	0 [-5]
<b>Total</b>	<b>100</b>