**CSE-271: Object-Oriented Programming**

**Exercise #4**

Max Points: 20

|  |  |
| --- | --- |
| **Name:** | Jacob Igel |

|  |  |
| --- | --- |
| Pin | For your own convenient reference – You should first save/rename this document using the naming convention **MUid\_Exercise4.docx** (example: raodm\_Exercise4.docx) prior to proceeding with this exercise. |

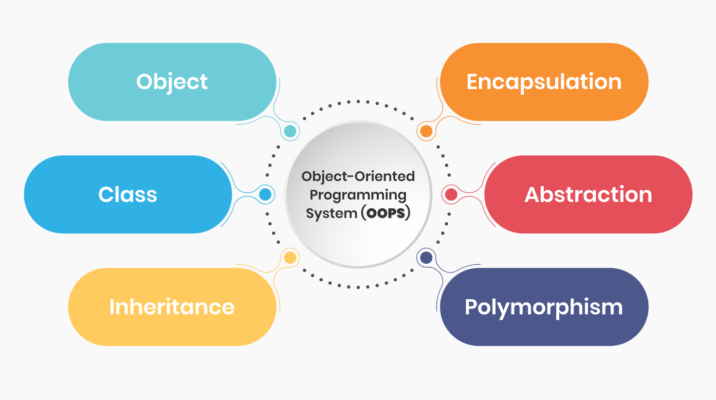
|  |
| --- |
| **Objectives**: The objectives of this exercise are to:   1. Explore the concepts of inheritance 2. Experiment concepts of polymorphic method calls 3. Gain familiarity with Java’s approach for inheritance and polymorphism 4. Developing Java classes with constructors, getter, and setter method(s) 5. Overriding special methods such as: toString and equals   Fill in answers to all of the questions. For some of the questions you can simply copy-paste appropriate text from Eclipse output into this document. You may discuss the questions or seek help from your neighbor, TA, and/or your instructor. |

# Part #0: One time setup of Eclipse (IDE) – Only if needed

|  |  |
| --- | --- |
| Eclipse Logo A2 by dj-fahr on DeviantArt | We already configured Eclipse’s source formatter and Checkstyle plug-in as part of Lab #1. If your Eclipse is not configured (because you are using a different computer) then use the instructions from Lab #1 to configure Eclipse. |

# Part #1: Generic concepts of Object-oriented programming

*Estimate time: < 15 minutes*



**Background**: Object-oriented Programming (OOP) is a programming paradigm that is widely used and adopted by several mainstream programming languages, such as: C++, C#, JavaScript, Java, and Python. Hence, understanding the generic concepts underlying OOP is very important for your future careers, immaterial of the programming-language that you may be working with. Moreover, clearly and concisely explaining concepts is a very important skills for your future job-interviews and in your jobs. Hence, the exams also involve such questions, and in the labs, we will practice this style of questions.

**Exercise**: Briefly (2-to-3 sentences each) respond to the following questions regarding generic concepts of object-oriented programming (OOP).

1. List the 4 tenants (or principles) of OOP? Just list the 4 words/terms and that is sufficient for this specific question in this specific lab (obviously, in job situations you should elaborate and provide good examples)

|  |
| --- |
| 1. Encapsulation 2. Abstraction 3. Inheritance 4. Polymorphism |

1. Briefly describe (2-to-3 sentences max) the concept of “inheritance” and how it is achieved in Java?

|  |
| --- |
| Inheritance is used when a class inherits attributes from another class or its parent class. We are able to reuse some code of existing super classes. Child class can override methods to perform different operations |

1. Briefly describe (2-to-3 sentences max) the concept of “polymorphism” and how it is achieved in Java?

|  |
| --- |
| Polymorphism is an inheriting-class that can override (different from overload) methods to perform different operations. This can come in two forms: static and dynamic. |

# Part #2: Inheritance and polymorphism in Java

*Estimated time: < 25 minutes*

**Background**: The generic OOP concepts (covered in previous part) are supported by different object-oriented programming-languages in slightly different ways. Java has a specific approach for supporting the 4-key tenants of OOP using Java classes. However, similar syntax and semantics also apply to other object-oriented programming-languages.

**Exercise**: Briefly (2-to-3 sentences each) respond to the following questions regarding Java classes.

1. Briefly (2-to-3 sentences) describe the difference between the access modifiers private and protected.

|  |
| --- |
| Private modifiers mean that only methods within the same class can access them. Protected modifiers mean that any accessible descendant and any method within the package (within or outside the package) can access them. |

1. What is the parent class of the Person class shown in the adjacent figure?

|  |  |
| --- | --- |
| Java.lang.Object | public class **Person** {  // Stuff here…  } |

1. ~~What is the output generated by the main() method shown below:~~

|  |  |  |
| --- | --- | --- |
| **public** **class** A {  **public** **int** getOne() {  **return** 1;  }  **public** **static** **int** getTwo() {  **return** 2;  }  } | **public** **class** B **extends** A {  **public** **int** getOne() {  **return** 3;  }  **public** **static** **int** getTwo() {  **return** 4;  }  } | **public** **static** **void**  main(String [] args) {  B b = **new** B();  A a = (A) b;  System.out.println(a.getOne());  System.out.println(a.getTwo());  } |

|  |  |
| --- | --- |
| ~~The output from the main method is:~~ |  |

~~Briefly describe how you arrived at the above output:~~

|  |
| --- |
|  |

1. Recollect that Java provides two features to minimize code duplication for constructors.
2. The first one is to call a constructor in the base class via the super keyword.
3. Calling a different constructor in the same class via this keyword.

Using the above two approaches rewrite class B to be more concise by minimizing redundant code.

|  |  |  |
| --- | --- | --- |
| **public** **class** A {  **private** **int** age;  **private** String name;  **public** A(**int** age,  String name) {  setAge(age);  setName(name);  }  **void** setAge(**int** age) {  **this**.age = age;  }    **void** setName(String name) {  **this**.name = name;  }  } | **public** **class** B **extends** A {  **private** String id;  **public** B() {  setAge(-1);  setName("invalid");  setID("-1");  }    **public** B(**int** age,  String name) {  setAge(age);  setName(name);  setID("-1");  }    **public** B(**int** age,  String name,  String id) {  setAge(age);  setName(name);  setID(id);  }  **void** setID(String id) {  **this**.id = id;  }  } | **pblic class B extends A {   private String id;     public B() {   this(-1, "invalid", "-1");   }     public B(int age, String name) {   this(age, name, "-1");   }     public B(int age, String name, String id) {   super(age, name);   setID(id);   }     void setID(String id) {   this.id = id;   }  }**  **}** |
|  |  |  |

|  |
| --- |
| **public** **class** Car {  **private** String make;  **private** **int** mileage;  } |

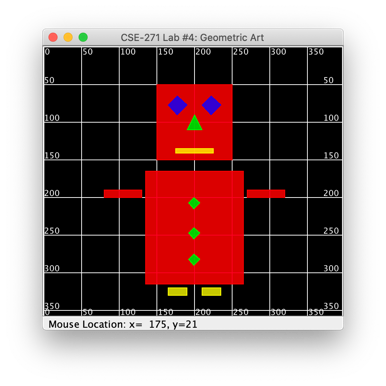
1. ~~Implement the equals method for the adjacent Car class such that two objects are equal only if all of the instance variables have the same values.~~

|  |
| --- |
| Public class Car {  @Override  Public Boolean equals(object obj) {  If(obj == null || car.class != obj.getClass()) {  Return false;  }  Car other = (Car) obj  Return(car.make.equals(this.make) && car.mileage == this.mileage);  }  } |

# Part #3: Develop hierarchy of Java classes via Inheritance

*Estimated time: 30 minutes*

**Background**: A key advantage of inheritance is that it reduces the amount of object-oriented code to be developed. Hence, designing good class hierarchies that enable effective code reuse is central to OOP – remember, “*less is more*”.



**Exercise**: In this part of the exercise, you are given a set of independent shape classes, namely: Rectangle, Triangle, Rhombus. These classes are used by GeometricArt class to display art as shown in the adjacent screenshot. Your task is to use inheritance to minimize redundant code in the 3 shapes by building a class hierarchy using the supplied Shape class the base class. Of course, you need to setup a Java project in Eclipse, download the starter code, and make the necessary modifications.

In order to pass the test on Code, the Rhombus class should extend the Rectangle class.

**Testing**:

Once you have completed the implementation your class hierarchy, your output should appear as shown in the adjacent screenshot. That is, you will be preserving the same features/functionality, but with less code.

# Part #4: Work with polymorphic methods

*Estimated time: 30 minutes*

**Background**: In OOP, polymorphism refers to the ability for a child class to override methods in the parent class, thereby performing different operations even when the method is called via a parent class reference. In Java, it is accomplished by permitting child-class to override methods and then just using parent-class references along with polymorphic method calls. This enables reusing code with little-to-no modifications.

**Exercise**: In this part of the exercise, you will be implementing the 3 methods in the ShapeAnalyzer class. Refer to the Javadoc for each method on what operations the methods should perform and suitably implement them using polymorphic calls to the getArea method in the Shape class.

**Testing**: Once you have implemented the 3 methods in the ShapeAnalyzer, run the program and the expected output is:

|  |
| --- |
| Total area: 33425.0  Largest shape's area: 19500.0 |

# Part #5: Submit to Canvas via CODE plug-in

*Estimated time: < 5 minutes*

**Exercise:** You will be submitting the following files via the Canvas CODE plug-in:

1. This MS-Word document saved as a PDF file – **Only submit PDF file**.
2. The Java source files: Shape.java, Rectangle.java, Triangle.java, Rhombus.java, and ShapeAnalyzer.java that you developed in this exercise.

Ensure you actually complete the submission on Canvas by verifying your submission