**CSCI 2302**

**Inheritance & Polymorphism Chapter**

**Inheritance & Polymorphism Lab with Geometric Objects**

Contents

[Intro: 1](#_Toc98307969)

[Learning Goals 1](#_Toc98307970)

[Notes 1](#_Toc98307971)

[Task 4](#_Toc98307972)

[Submit 7](#_Toc98307973)

# Intro:

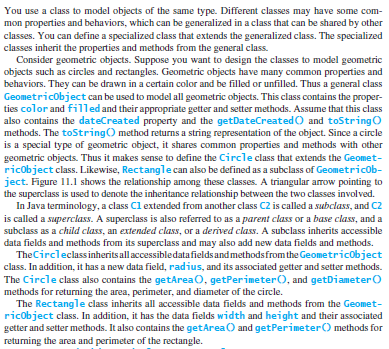


Inheritance and polymorphism are pillars of OOP. Inheritance describes a relationship between two classes. The relationship is defined as is-a. For example, a PineTree is a type of Flora, or a Circle is a GeometricObject. Inheritance means to define a new class from an existing class. This means you, as a programmer can reuse existing code (a class that has already been defined). What does that mean for the object that you want to define using the existing code? It means that the new object will have all of the states and all of the behaviors of the existing class. To this, you will define its own states and behaviors.

Learning Goals: To demonstrate how inheritance is implemented in a Java program. To compare how this class relationship, is-a relationship, differs with aggregation, has-a relationship. So that in the future we can select which class relationship is needed as we build our own program systems. To understand and apply overridden methods and constructor chaining.

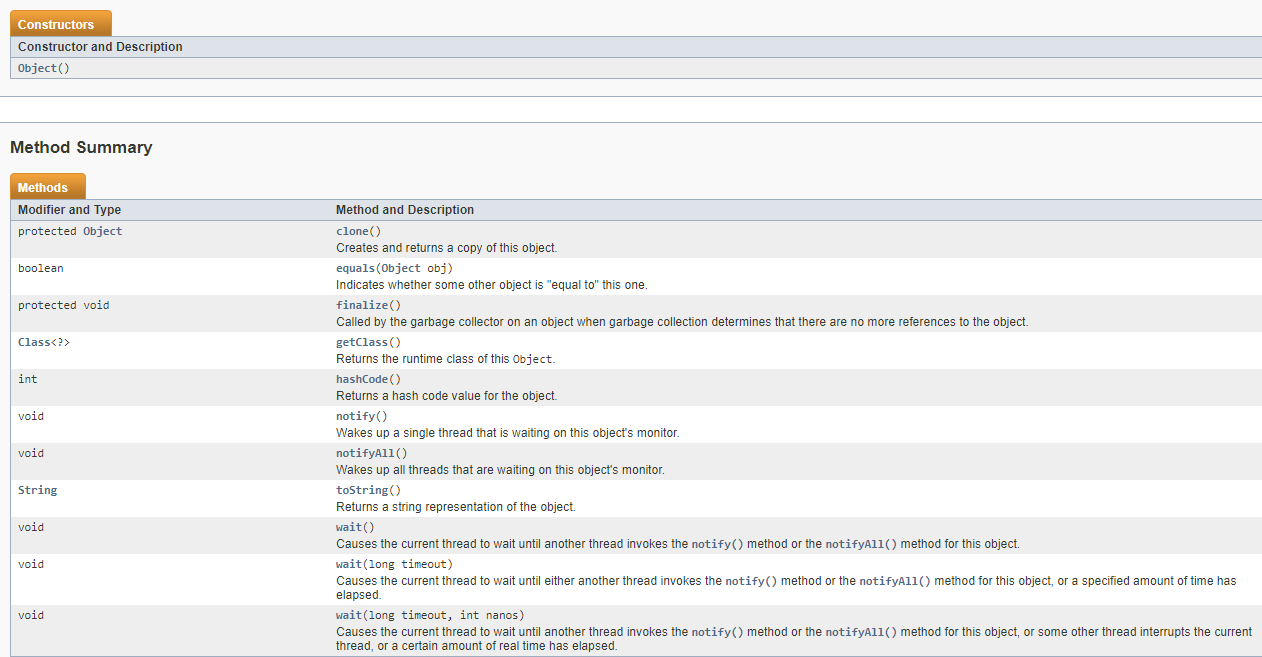
Notes:

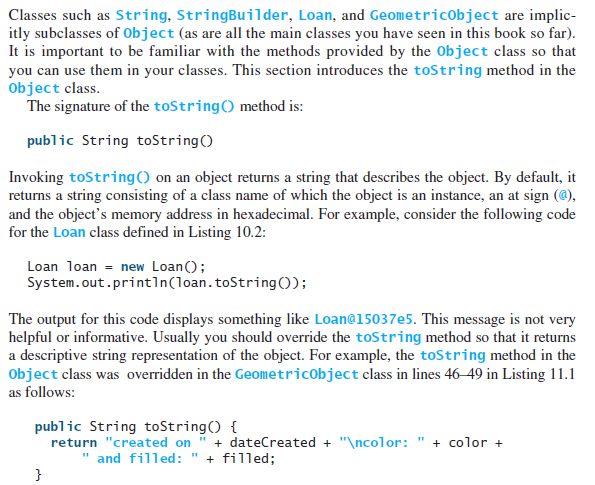


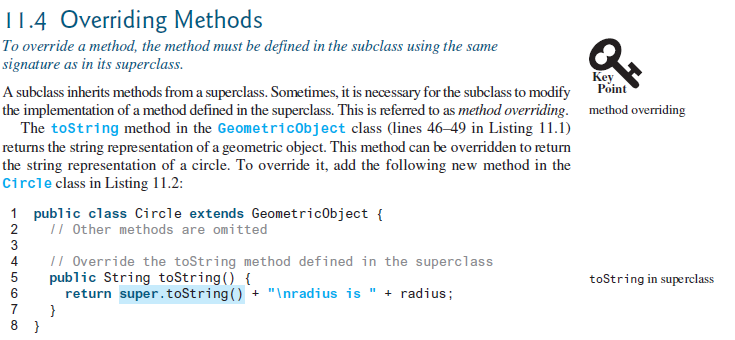


Every time you define a class, it is inheriting from the Object class that is pre-defined in Java. This means that it has everything that the Object class has.

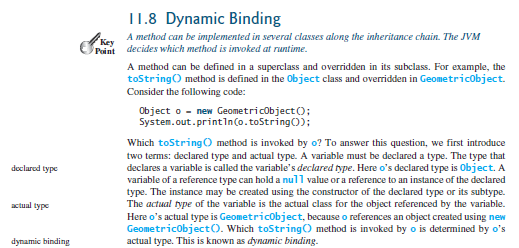


https://docs.oracle.com/javase/7/docs/api/java/lang/Object.html 





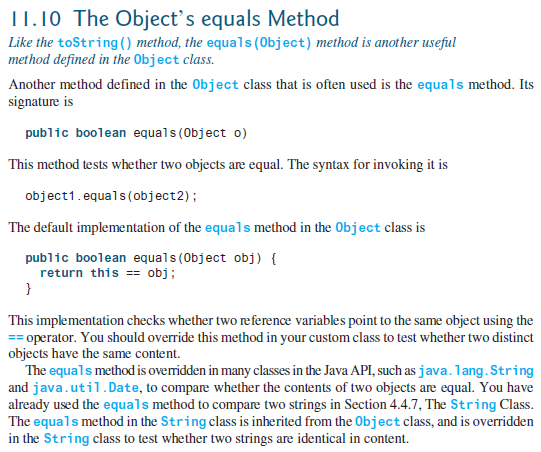
Since methods with the same signature can be defined anywhere along the inheritance chain, how does the JVM know which method to go to? This is accomplished with dynamic binding. And it helps to use @Override when the methods are overridden. Dynamic binding also allows polymorphism.



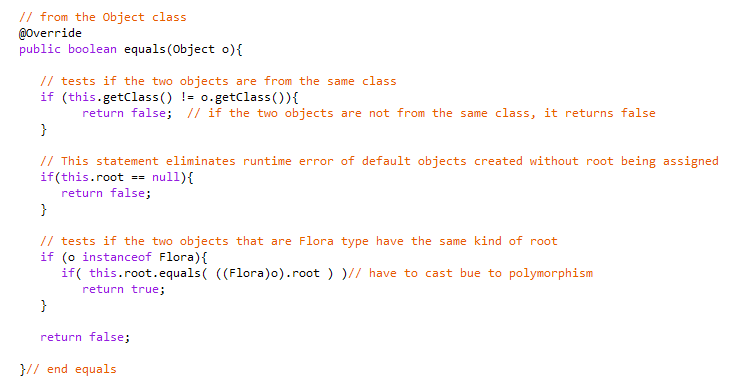
Task:

1. Download TestGeoObj.java. Rename TestGeoObj.java to mysfasuUserName\_TestGeoObj.java and complete the following steps. If any step cannot be accomplished, comment that step out and write a comment why that cannot be done.

1. In the Triangle class, implement/define an equals method, which will compare the area of two triangles to determine if the two triangles have the same area.



Another example, from the InheritanceAndPolymorphismChapterLearning, Flora class:

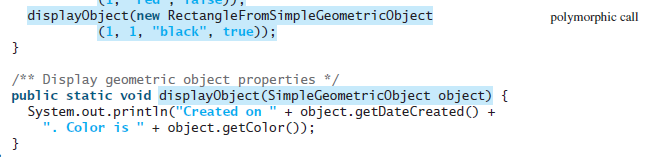


1. In the SquareWithToStringOverride, implement/define the toString method.

Complete the following in the mysfasuUserName\_TestGeoObj.java file:

1. Declare and assign/Instantiate a GeometricObject object with purple and true for its states.
2. Declare and assign/Instantiate a Triangle object with grey, false, 2, 2, and 2 for its states.
3. Declare and assign/Instantiate a Square object with white, true, and 3 for its states.
4. Declare and assign/Instantiate a SquareWithToStringOverride with silver, false, and 8 for its states.
5. Declare and assign/Instantiate a new GeometricObject object with a SquareWithToStringOverride and black, true, and 5 for its states.
6. Declare and assign/Instantiate a new SquareWithToStringOverride object with GeometricObject object with red and false for its state.
7. Declare and assign/Instantiate an Object object with nothing for its states.
8. Define a method that accepts Objects and invoke the toString method in the method.

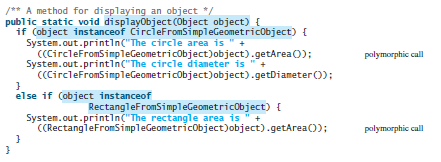
The inheritance relationship enables a subclass to inherit features from its superclass with additional new features. A subclass is a specialization of its superclass; every instance of a subclass is also an instance of its superclass, but not vice versa. For example, every Circle is a geometricObject, but not every geometricObject is a Circle. Therefore, you can always pass an instance of a subclass to a parameter of its superclass type.

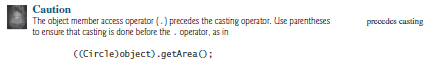


And you can always access the states/fields or behaviors/methods of the super class.

1. Make each object from steps 4 – 10 invoke the method created in step 11.
2. Define a method that accepts Objects and will invoke the getArea method for each object type.

Since you can always get to the information of the super type/declared type, how do you get to the information of the subtype/actual type? You will have to cast the object to the subtype/actual type. It is a good practice to ensure that the object is an instance of another object before attempting a casting. This can be accomplished by using the instanceof operator.





1. Make each object from steps 4 – 10 invoke the method created in step 13.
2. Declare and assign/Instantiate a Triangle object with black, true, 2, 2, and 2 for its states.
3. Declare and assign/Instantiate a Triangle object with purple, true, 2, 1, and 2 for its states.
4. Invoke the equals method (defined/implemented in step 2) twice:
   1. Once with the Triangle from step 5 to the Triangle from step 15
   2. Once with the Triangle from step 5 to the Triangle from step 16

Submit: Submit the completed program file in the Dropbox in Brightspace by D2L.