**CSCI 2302**

**Inheritance & Polymorphism Chapter**

**Fruit Objects Lab**

Contents

[Intro 1](#_Toc84500282)

[Notes 1](#_Toc84500283)

[Learning Goals 3](#_Toc84500284)

[Task 3](#_Toc84500285)

[Submit 8](#_Toc84500286)

Intro: Inheritance and polymorphism are pillars of OOP. Inheritance describes a relationship between two classes. The relationship is defined as is-a. Every time you define a class, it is inheriting from the Object class that is pre-defined in Java, because a class is a ADT/an object.

Polymorphism means that a variable of a supertype (the declared type) can refer to a subtype (the actual) object. When you have a polymorphic variable, it is always possible to access the supertype/declared type states and behaviors. To access the subtype/actual type, you must first test that the object is the subtype/actual type, and then cast the variable to the subtype/actual type – pay attention to the parenthesis.

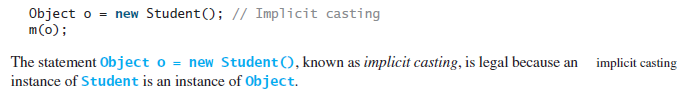
Notes:

When you are using inheritance and polymorphism, you will have to understand the difference between **static** binding and **dynamic** binding.

Static binding occurs during compile time. The compiler is looking to match up the invoking statements with meathod headers. If the complier finds the matches, then the program compiles (ignoring the other syntax errors here for simplicity sakes), if the compiler cannot find matches, then the program does not compile.

Dynamic binding occurs during runtime. It is the JVM that is looking for the matches. The issues to think about are inheritance and polymorphism. Since any ADTs/objects/classes are automatically inherited from the Object class, and can have an inheritance chain, the issue then is where to find the matching invoking statement to the method header. As a programmer, you use identifiers that accurately describe what a method does, and can reuse that identifier in different classes.

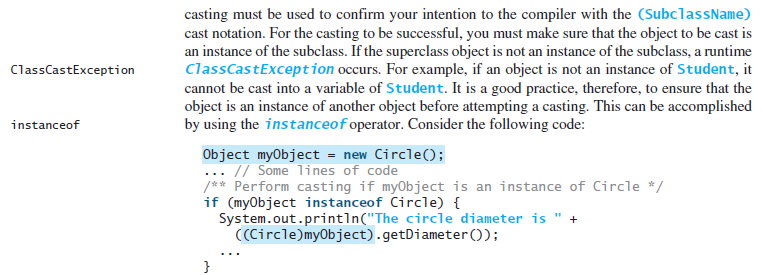
When you have a polymorphic variable, it is always possible to access the supertype/declared type states and behaviors. To access the subtype/actual type, you must first test that the object is the subtype/actual type, then cast the variable to the subtype/actual type – pay attention to the parenthesis.



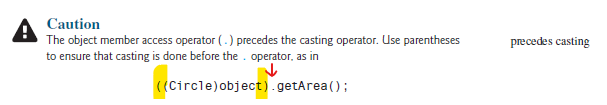
Even though you can see that o is really a Student object, the compiler is not clever enough to know it. To tell the compiler that o is a Student object, use **explicit** casting.

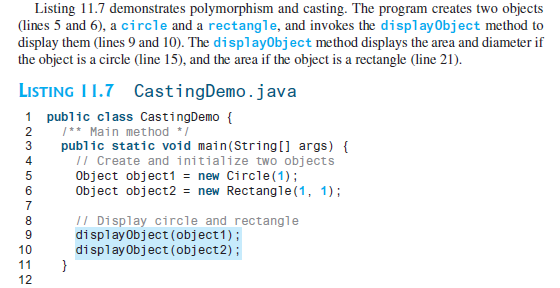


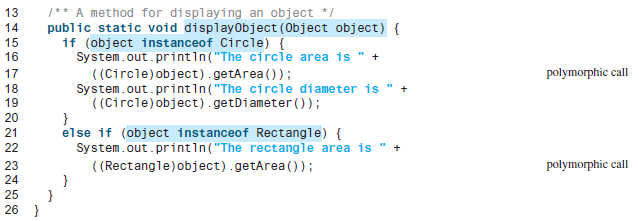












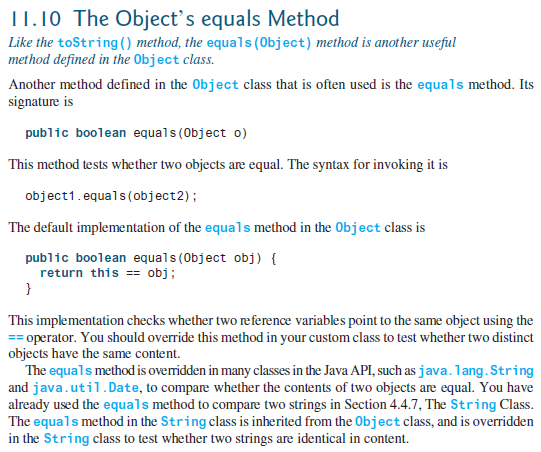
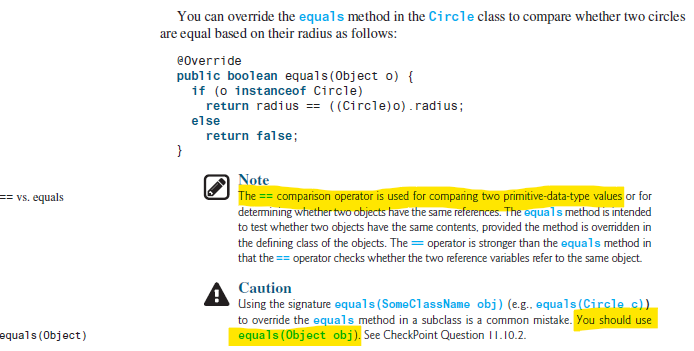
Learning Goals: To understand inheritance and polymorphism, to correlate what it means for an object to be a is-a relationship to another object, to illustrate how that relationship can be used in polymorphism, to implement how to access the common states/fields and behaviors/methods, and to implement how to access the subtype/actual type states/fields and behaviors/methods.

Task: Complete the steps outlined below.

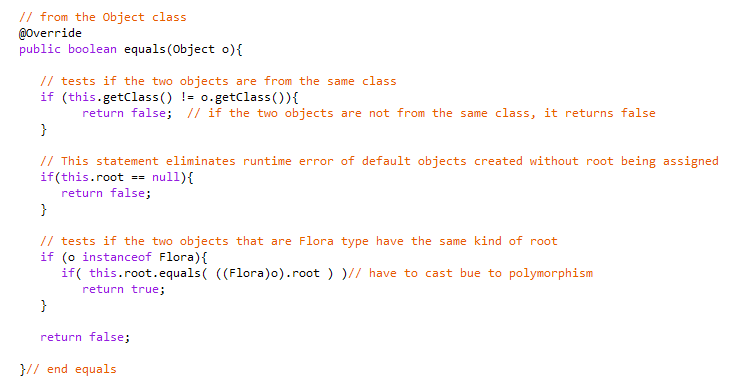
Complete the following using the following UML diagram.

1. Define/Implement the Fruit object.
2. Define/Implement the Apple object.
3. Define/Implement the GoldenDelicious object.
4. Define/Implement the mysfaUsername\_InheritPolyTestFruit class to do the following:
   1. Instantiate a Fruit object, with “fruit” for the name field/state value
   2. Instantiate an Apple object, with “apple” for the name field/state value
   3. Instantiate a GoldenDelicious object, with “Golden Delicious” for the name field/state value
   4. Instantiate a Fruit object that refers to an Apple, with “apple” for the name field/state value
   5. Instantiate an Apple object that refers to a GoldenDelicious, with “Golden Delicious” for the name field/state value
   6. Instantiate an Object object
   7. Define the printFruit method which prints the toString method
   8. Invoke the printFruit method using all the variables from a – f
   9. Define the printName method which prints the name field/state
   10. Invoke the printName method using all the variables from a – f

Bonus Points: Define the equals method in the Fruit class which compares the name state/field values and returns true if the values are the same. Ensure that it works by invoking this method at least two (2) times.

Another example, from the InheritanceAndPolymorphismChapterLearning, Flora class:



|  |
| --- |
| **InheritPolyTestFruit** |
| - fruit: Fruit  -apple: Apple  -gd: GoldenDelicious  - fruitThatIsApple: Fruit  -appleThatIsGD: Apple  - obj: Object |
| +main(): void  +printFruit(o Object): void  +printName(o Object): void |

|  |
| --- |
| **Fruit** |
| -name: String  -type: String = “Fruit object” |
| +Fruit(name: String)  +getName(): String  +toString(): String |

|  |
| --- |
| **Apple** |
|  |
| +Apple(name: String)  +toString(): String |

|  |
| --- |
| **GoldenDelicious** |
|  |
| +GoldenDelicious(name: String)  +toString(): String |

Submit: The Fruit.java, Apple.java, GoldenDelicious.java, and mysfaUsername\_InheritPolyTestFruit.java files, or define all the classes/objects/ADTs in the mysfaUsername\_InheritPolyTestFruit.java and submit just that file.