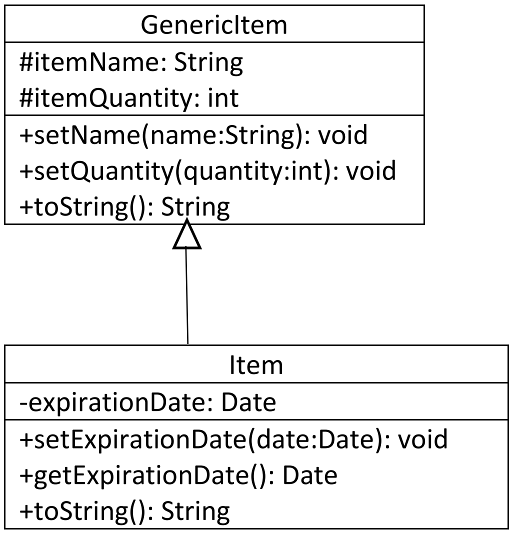
CS163/164 – Polymorphism Worksheet

Name:____

<u>Inheritance</u>

Inheritance is a king of 'is-a' relationship. Classes inherit from other classes, gain their properties (data + methods).

UML (Unified Modelling Language) Derived Class Example


GenericItem
#itemName: String
#itemQuantity: int
+setName(name:String): void
+setQuantity(quantity:int): void
+toString(): String
Item
-expirationDate: Date
+setExpirationDate(date:Date): void
+getExpirationDate(): Date
+toString(): String

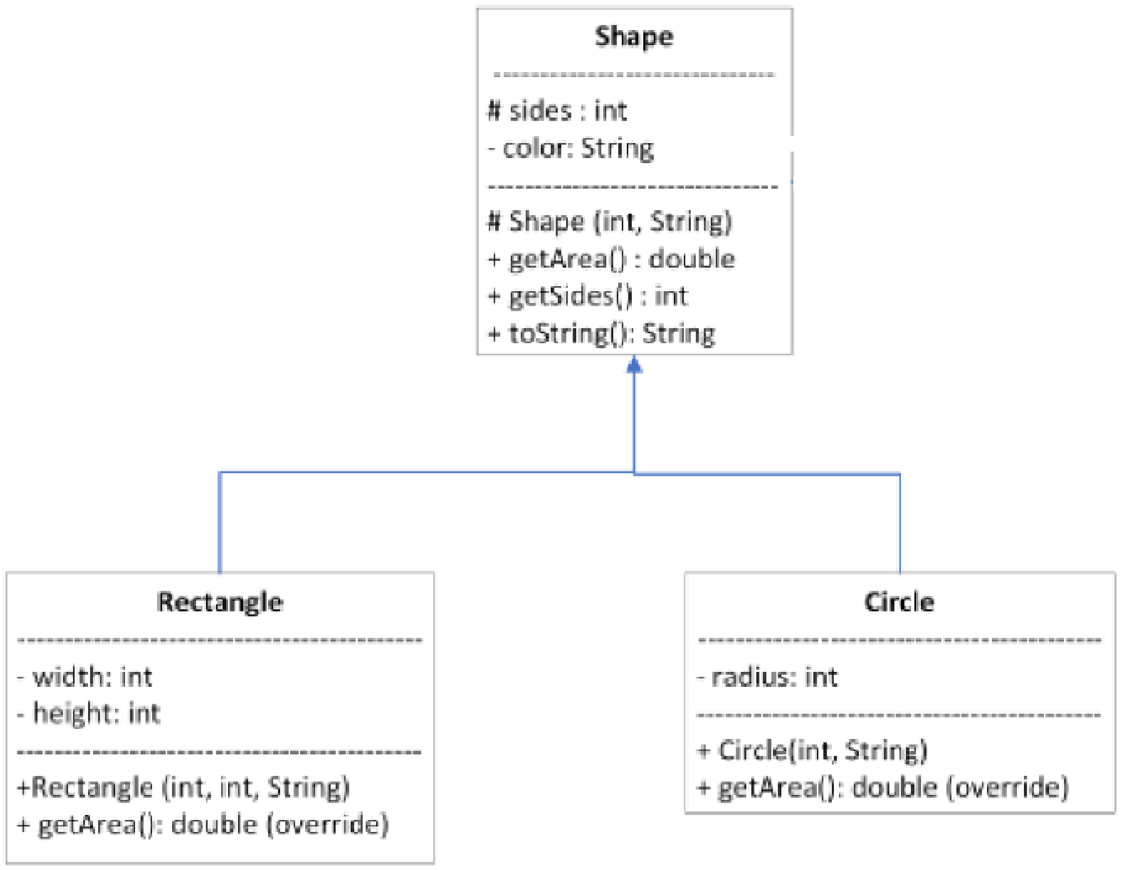
Member Access

- private
# protected
+ public

<u>Polymorphism</u>

Refers to determining which program behavior to execute depending on data types. Method overloading is a form of compile-time polymorphism wherein the compiler determines which of several identically-named methods to call based on the method's arguments. Another form is runtime polymorphism wherein the compiler cannot make the determination but instead the determination is made while the program is running. One scenario requiring runtime polymorphism involves derived classes. Programmers commonly create a collection of objects of both base and derived class types. Ex: the statement ArrayList<GenericItem> inventoryList = new ArrayList<GenericItem>(); declares an ArrayList that can contain references to objects of type GenericItem or ProduceItem. ProduceItem derives from GenericItem.

Considers the following UML diagram to answer the following questions:



1. Mark T (True) of F (False) the following statements:
   (a) Shape is a super class
   (b) Rectangle is a super class
   (c) Circle is a sub class

2. For each line of code below, indicate if that is correct or incorrect. Justify your answer.

```
ArrayList<Shape> listShape = new ArrayList<>();
Shape s1 = new Shape(2, "green");
Rectangle r1 = new Rectangle(4,2, "red");
Circle c1 = new Circle(3, "blue");
listShape.add(s1);
listShape.add(r1);
listShape.add(c1);
for(Shape s: listShape){
    System.out.println(s);
}
```

3. Are the following statements, correct? Justify your answer.

```
Rectangle r2 = new Shape(3, "black");
Shape s2 = new Rectangle(12, 2, "gray");
Shape s3 = new Circle(4, "yellow");
```

4. Consider the following implementation for class Shape to implement the classes Rectangle and Circle.

```java
public class Shape {
    protected int sides;
    private String color;

    public Shape(int sides, String color){
        this.sides = sides;
        this.color = color;
    }
    public int getSides(){
        return sides;
    }
    public String getColor(){
        return color;
    }
    public String toString(){
        return "Sides: " + sides + " Color: " + color;
    }
}
```

5. Implement the toString method for Rectangle and Circle. Implement the method getPerimeter to the Circle class.
6. Write a main method that:
    a. Creates an ArrayList of Shapes
    b. Add different Shapes objects into the ArrayList, you should add at least one of each class.
    c. Loop through your ArrayList and print all the objects using toString. You need to print the class of which that object belongs before your toString.
    You will need to use instanceof to know which class that object belongs.
    For example, if would like to print only the Rectangle objects in the ArrayList shapes you could do something like this:

    ```
    for(Shape s : shapes) {
      if(s instanceof Rectangle) System.out.println(s);
    }
    ```

    Casting between Objects

    - Tell the compiler you know what you are doing
    - Forces the compiler to treat an object like another object
    - ONLY valid if going between super and subclasses

    ```java
    Shape s3 = new Circle(4, "yellow");
    if(s3 instanceof Circle){
            Circle c3 = (Circle) s3;
            System.out.println(c3.getPerimeter());
    }
    ```