A document on UML Class diagram

1. The Fan Class. Design a class named Fan to represent a fan. the class contains:

Three constants named SLOW, MEDIUM and FAST with values 1, 2 and 3 to denote the fan speed.

1. An int data field named speed that specifies the speed of the fan (default SLOW)
2. A boolean data field named on that specifies whether the fan is on (default false)
3. A double data field named radius that specifies radius of the fan (default 5)
4. A string data field named radius that specifies the color of the fan (default white)
5. A no-arg constructor that creates a default fan
6. The accessor and mutator methods for all four data fields
7. A method named toString() that returns a string description for the fan. If the fan is on, the method returns the fan speed, color and radius in one combined string. If the fan is not on, the method returns fan color and radius along with the string "fan is off" in one combined string.

Fan	
+SLOW = 1	Constant.
+MEDIUM = 2	Constant.
+FAST = 3	Constant.
-speed: int	The speed of this fan (default 1).
-on: boolean	Indicates whether the fan is on (default false).
-radius: double	The radius of this fan (default 5).
-color: String	The color of this fan (default white).
+Fan()	Constructs a fan with default values.
+getSpeed(): int	Returns the speed of this fan.
+setSpeed(speed: int): void	Sets a new speed for this fan.
+isOn(): boolean	Returns true if this fan is on.
+setOn(on: boolean): void	Sets this fan on to true or false.
+getRadius(): double	Returns the radius of this fan.
+setRadius(radius: double): void	Sets a new radius for this fan.
+getColor(): String	Returns the color of this fan.
+setColor(color: String): void	Sets a new color for this fan.
+toString(): String	Returns a string representation for this fan.

2. (The Stock class) Design a class named Stock that contains:

1. A string data field named symbol for stock's symbol.
2. A string data field named name for the stock's name.

3. A double data field named previousClosingPrice that stores the stock price for the previous day.
4. A double datafield named CurrentPrice that stores the stock price for the current time.
5. A constructor that creates a stock with specified symbol and name.
6. The accessor methods for all data fields.
7. The mutator methods for previousClosingPrice and currentPrice.
8. A method named getChangePercent() that returns the percentage changed from previousClosingPrice.

Draw the UML diagram for the class. Implement the class. Write a test program that creates a Stock object with the stock symbol SUNW, the name SUN Microsystems Inc, and the previous closing price of 100. Set a new current price to 90 and display the price-change percentage.

Stock	
-symbol: String	The symbol of this stock.
-name: String	The name of this stock.
-previousClosingPrice: double	The previous closing price of this stock.
-currentPrice: double	The current price of this stock.
+Stock(symbol: String, name: String)	Constructs a stock with a specified symbol and a name.
+getSymbol():String	Returns the symbol of this stock.
+getName():String	Returns the name of this stock.
+getPreviousClosingPrice(): double	Returns the previous closing price of this stock.
+getCurrentPrice(): double	Returns the current price of this stock.
+setPreviousClosingPrice(price: double): void	Sets the previous closing price of this stock.
+setCurrentPrice(price: double): void	Sets the current price of this stock.
+getChangePercent(): double	Returns the percentage of change of this stock.

3. The MyPoint class. Design a class named MyPoint to represent a point with X and Y-coordinates. The class contains:

1. Two data fields X and Y that represent the coordinates with get methods.
2. A no-arg constructor that creates a point (0, 0).
3. A constructor that constructs a point with specified coordinates.
4. To get methods for data fields X and Y, respectively.
5. A method named distance that returns the distance from this point to another point of the MyPoint type.
6. A method named distance that returns the distance from this point to another point with specified -X and -Y coordinates.

Draw UML diagram for the class. Implement the class. Write a test program that creates two points (0, 0) and (10, 30.5) and displays the distance between them.

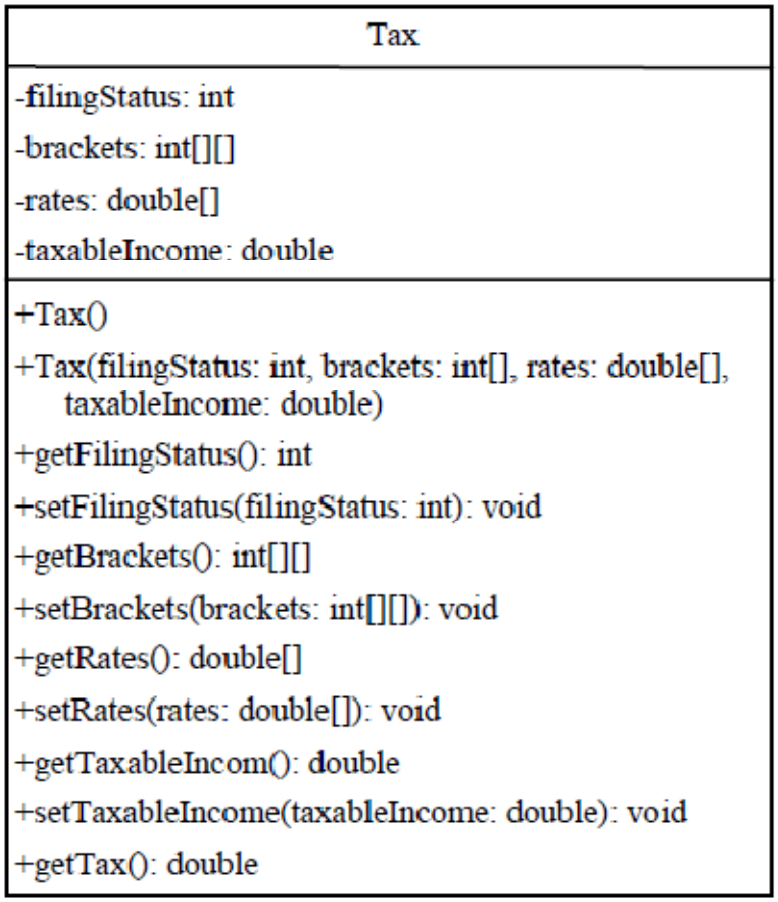
MyPoint	
-x: double	x-coordinate of this point.
-y: double	y-coordinate of this point.
+MyPoint()	Constructs a Point object at (0, 0).
+MyPoint(x: double, y: double)	Constructs an object with specified x and y values.
+getX(): double	Returns x value in this object.
+getY(): double	Returns y value in this object.
+distance(secondPoint: MyPoint): double	Returns the distance from this point to another point.
+distance(p1: Point, p2: MyPoint): double	Returns the distance between two points.

4. (financial: the Tax class) Design a class named Tax to contain the following instance data fields:

int filingStatus: One of the four tax filing status. 0-single filer, 1-married filing jointly, 2-married filing separately and 3-head of the house-hold. Use public static constants SINGLE_FILER (0), MARRIED_JOINTLY (1), MARRIED_SEPERATELY (2) and HEAD_OF_HOUSEHOLD (3) to represent the status.

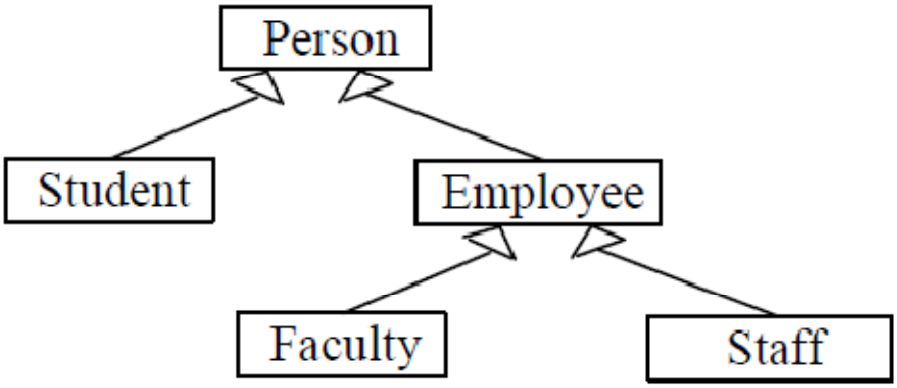
1. int[] [] brackets: Stores the tax brackets for each filing status.
2. double[] rates: Stores tax rates for each bracket.
3. double taxableIncome: Stores the taxable income.
4. Provide the get and set methods for each datafield and the getText() method that returns the tax. Also provide a no-arg constructor and the constructor Tax(filingStatus, brackets, rates, taxableIncome).

Draw the UML diagram for the class. Implement the class. Write a test program that uses the Tax class to print the 2001 and 2002 tax tables for taxable income from $50,000 to $60,000 with intervals of $1000 for all four statues. The tax rates for the year 2001-2002 you can assume.



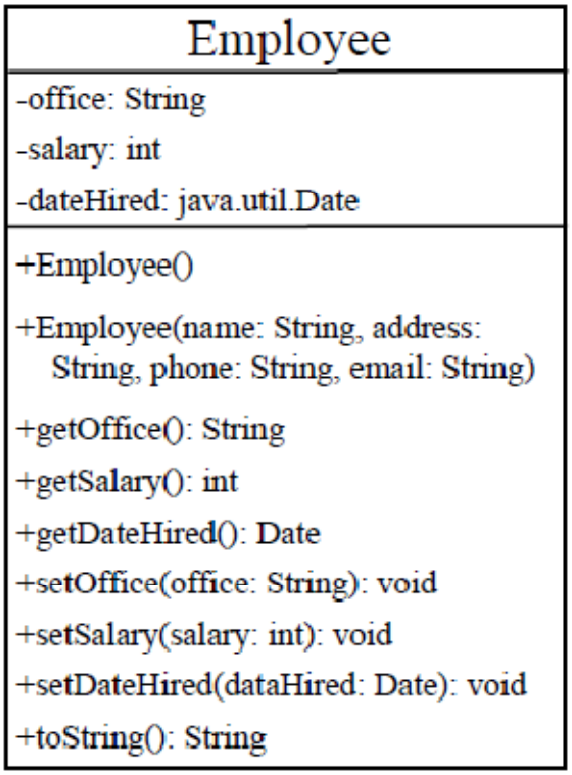
5. (The Person, Student, Employee, Faculty and Staff classes) Design a class named Person and its two subclasses named Student and Employee. Make Faculty and Staff subclass of Employee. A person has a name, address, phone number and email address. A student has a class status (Freshman, sophomore, junior or senior) Define the status as a constant. An employee has a office, salary and date-hired. Define a class named MyDate that contains the fields year, month and day. A faculty member has office hours and rank. A staff member has a title. Override toString method in each class to display the class name and the person's name.

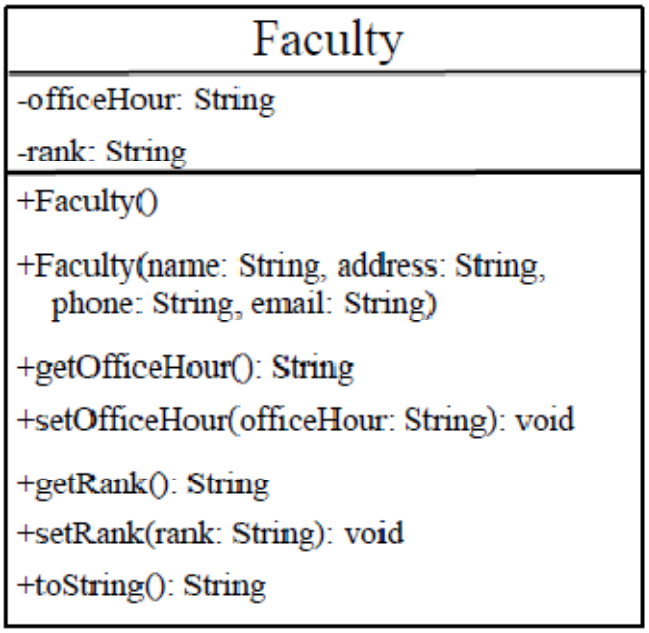
Draw the UML Classes. Implement the classes. Write a test program that creates a Person, Student, Employee, Faculty and Staff and invokes their toString methods.

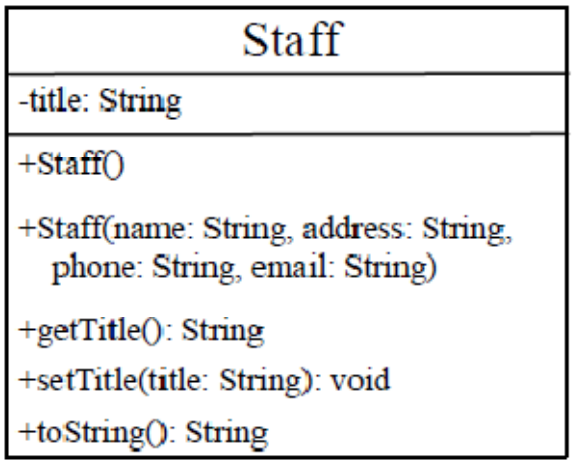


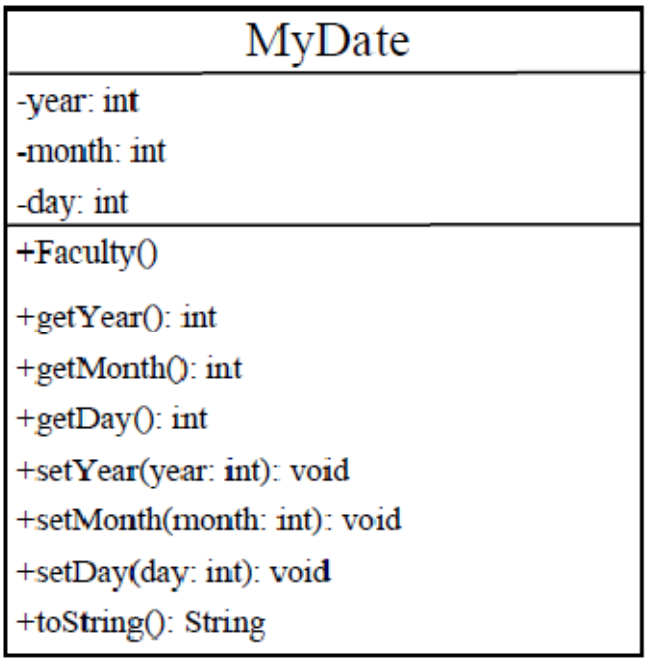
Person
-name: String -address: String -phone: String -email: String
+Person() +Person(name: String, address: String, phone: String, email: String) +getName(): String +getAddress(): String +getPhone(): String +getEmail(): String +setName(name: String): void +setAddress(address: String): void +setPhone(phonee: String): void +setEmail(email: String): void +toString(): String

Student
-status: String
+Student() +Student(name: String, address: String, phone: String, email: String) +getStatus(): String +setStatus(status: String): void +toString(): String

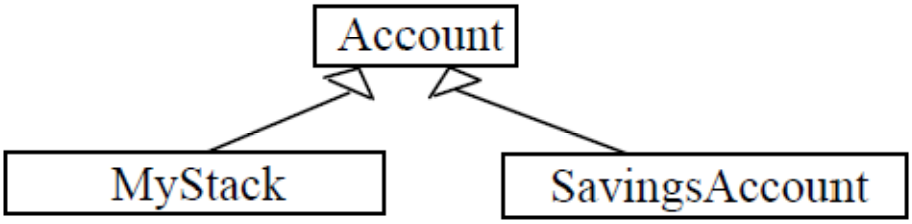


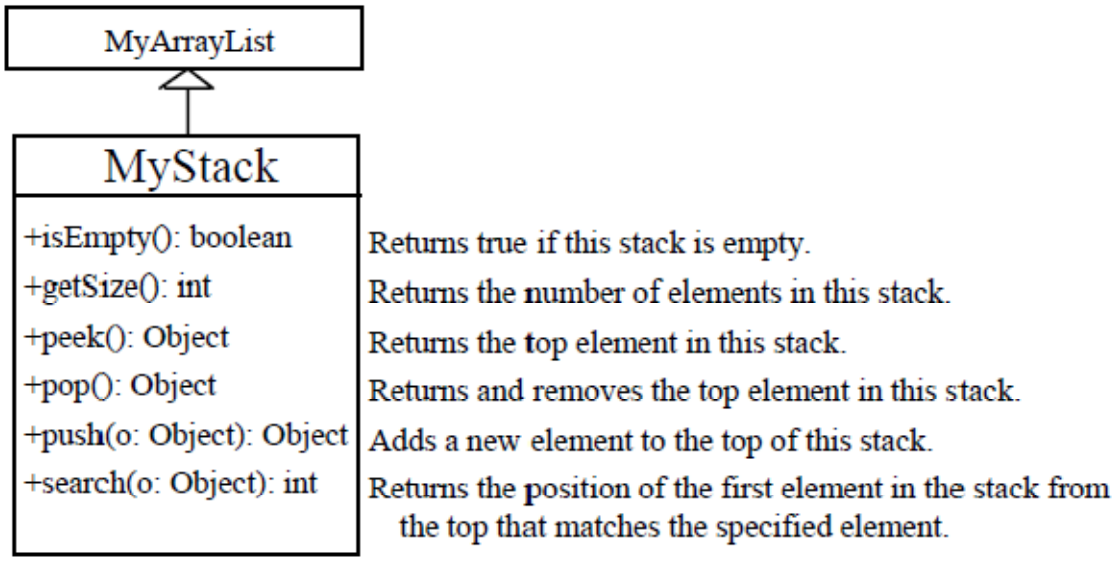




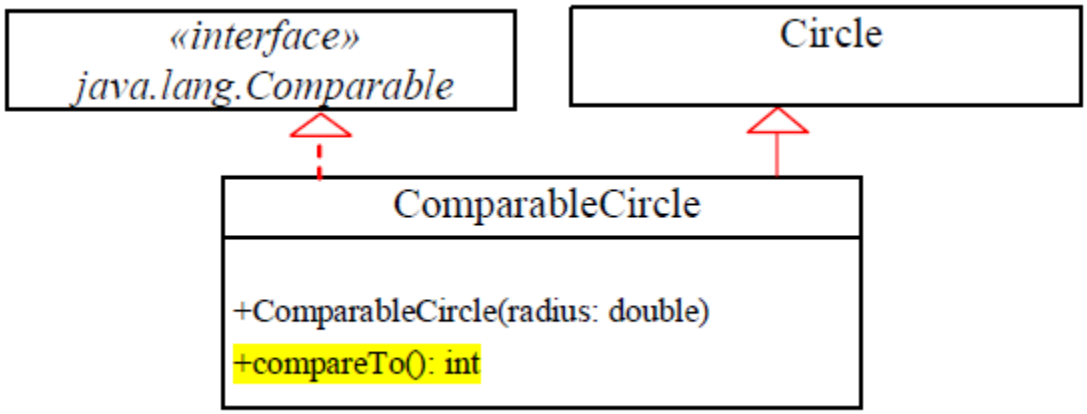


6. Try to evaluate following class diagram.



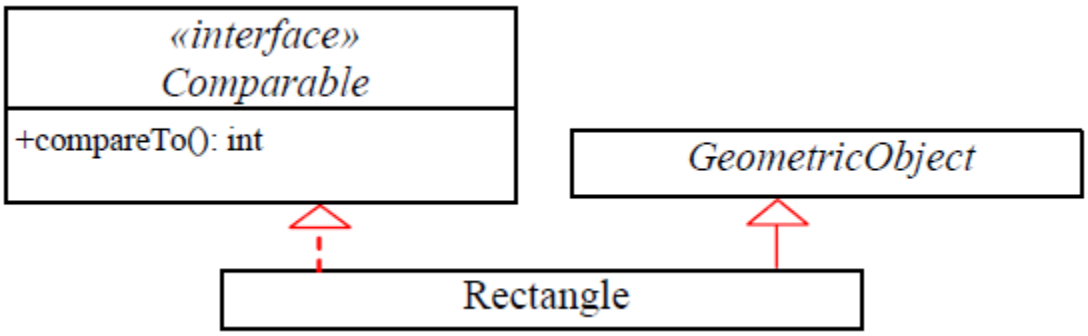


7. (The ComparableCircle class) Create a class ComparableCircle that extends Circle and implemets Comparable. Draw the UML diagram and implement the compareTo method to compare the circles on the basis of area. Write a test class to find the larger of two instances of ComparableCircle objects.



8. (Enabling Rectangle comparable) Write a Rectangle class to extend GeometricObject and implement the Comparable interface. Override the equals method in the Object class. Two Rectangle objects are

equal if their areas are the same. Draw the UML diagram that involves Rectangle, GeometricObject and Comparable.



9. (The Person and Student classes) Design the Student class that extends Person. Implement the compareTo method in Person class to compare person in alphabetical order of their last name. Implement the compareTo method to compare students in alphabetical order of their major and last name. Draw a UML diagram that involves Person, Student, Comparable and Name.