Lab 09
Inheritance & Polymorphism

P6 Solutions limited in scope to:		
• P1 Concepts • P2 Concepts • P3 Concepts • P4 Concepts • P5 Concepts	• Designing with Inheritance ○ super/sub class ○ is-a relationship ○ polymorphism ○ overriding methods	• Implementing with Inheritance ○ super keyword ○ abstract keyword ○ interface keyword ○ extends keyword ○ implements keyword

Submission Rules:

1. Submissions must be zipped into a **handin.zip** file. Each problem must be implemented in its own class file. Use the name of the problem as the class name.

2. You must use standard input and standard output for ALL your problems. It means that the input should be entered from the keyboard while the output will be displayed on the screen.

3. Your source code files should include a comment at the beginning including your name and that problem number/name.

4. The output of your solutions must be formatted exactly as the sample output to receive full credit for that submission.

5. Compile & test your solutions before submitting.

6. Each problem is worth up to 10 points total. The breakdown is as follows: 2 points for compiling, 3 points for correct output with sample inputs, 5 points for additional inputs.

7. This lab is worth a max total of: 40 points. You can complete as many problems as you like, but cannot receive more than 40 points towards the lab grade. All points in excess of that are for bragging rights. (Check the scoreboard to see how you did!)

8. Submission:

- You have unlimited submission attempts until the deadline passes
- You'll receive your lab grade immediately after submitting
- **IMPORTANT:** if your grade is lower than 70% when the deadline passes, then you must attend a recitation session & get TA signoff to receive full credit for that lab challenge.

Problem 1: Animal (10 points)

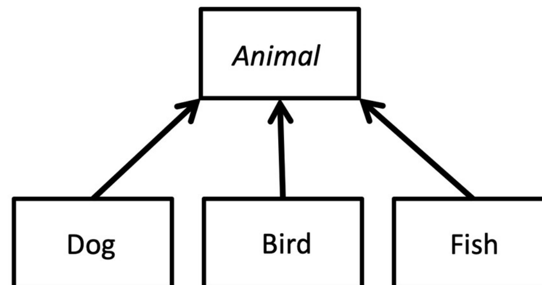
(Software Design) Create an abstract class that represents an Animal and contains abstract methods: move, call. Create concrete classes: Fish, Bird, Dog which extend the Animal class and implements those methods. A fish swims and calls "glub glub", a bird flies and calls "chirp chirp" , and a dog runs and calls "roof roof."

.

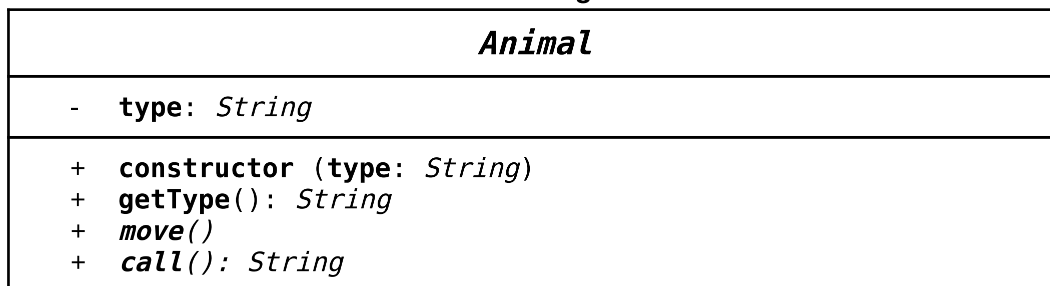
Software Architecture:

The **`Animal`** class is the abstract super class and must be instantiated by one of its concrete subclasses: Dog, Frog, or Fish, which extends from Animal.

UML Object Diagram



UML Class Diagram:



*Italicized class/method names within UML Class Diagram indicate abstract class/methods

Animal Constructor Summary:

Constructor	Description
`Animal(String type)`	Creates a Animal instance with a given type

Animal Method API:

Modifier and Type	Method and Description
`void`	**`getType`**`()` Returns the type of this animal
`abstract void`	**`move`**`()` moves this animal
`abstract String`	**`call`**`()` Returns as text representation the sound that this animal makes

UML Class Diagram:

Dog
+ **constructor** () + **move**() + **call***(): String*

*Italicized class/method names within UML Class Diagram indicate abstract class/methods

Dog Constructor Summary:

Constructor	Description
`Dog()`	Creates a Dog instance with "Dog" type

Dog Method API:

Modifier and Type	Method and Description
`void`	**move**() Displays to console message: this animal's type + " runs"
`String`	**call**() Returns as text "roof roof"

UML Class Diagram:

Bird
+ **constructor** () + **move**() + **call***(): String*

*Italicized class/method names within UML Class Diagram indicate abstract class/methods

Bird Constructor Summary:

Constructor	Description
`Bird()`	Creates a Bird instance with "Bird" type

Bird Method API:

Modifier and Type	Method and Description
`void`	**move**() Displays to console message: this animal's type + " flies"
`String`	**call**() Returns as text "chirp chirp"

UML Class Diagram:

Fish
+ **constructor** () + **move**() + **call***(): String*

*Italicized class/method names within UML Class Diagram indicate abstract class/methods

Fish Constructor Summary:

Constructor	Description
`Fish()`	Creates a Fish instance with "Fish" type

Fish Method API:

Modifier and Type	Method and Description
`void`	**move**() Displays to console message: this animal's type + " swims"
`String`	**call**() Returns as text "glub glub"

Tester Files:

Use the *TestAnimal.java* file to test your implementation. Compare your results with the *TestAnimal.txt* file.

Sample Method Calls	Sample Method Results

`Animal[] zoo = {  new Dog(),`
`                   new Fish(),`
`                   new Bird()`
`                };`
 	
`for (Animal animal : zoo){`
`     animal.move();`
 	
`System.out.println( animal.call() );`
`}`
	`"Dog runs\n"`
	`"roof roof\n"`
	`"Fish swims\n"`
	`"glub glub\n"`
	`"Bird flies\n"`
	`"chirp chirp\n"`

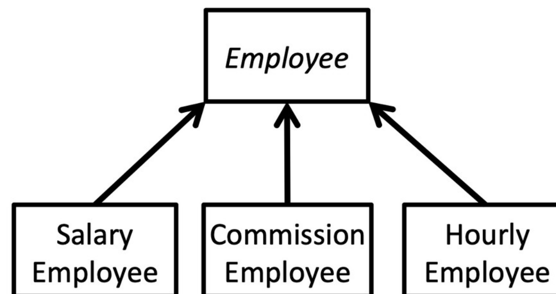
Problem 2: Employee (10 points)

(Software Design) Create an abstract class that represents an Employee and contains abstract method: payment. Create concrete classes: SalaryEmployee, CommissionEmployee, HourlyEmployee which extend the Employee class and implements that abstract method. A Salary Employee has a salary, a Commision Employee has a commission rate and total sales, and Hourly Employee as an hourly rate and hours worked.

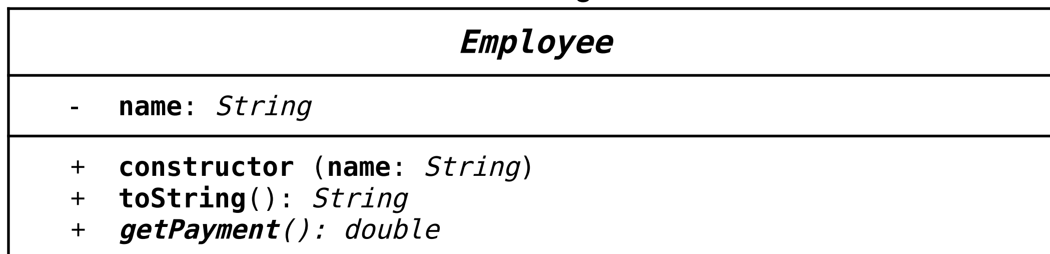
Software Architecture:

The **Employee** class is the abstract super class and must be instantiated by one of its concrete subclasses: SalaryEmployee, CommissionEmployee, or HourlyEmployee, which extends it.

UML Object Diagram



UML Class Diagram:



*Italicized class/method names within UML Class Diagram indicate abstract class/methods

Employee Constructor Summary:

Constructor	Description
`Employee(String name)`	Creates an Employee instance with a given name

Employee Method API:

Modifier and Type	Method and Description
`void`	**`toString`**`()` Returns a string containing the name of the employee
`abstract double`	**`getPayment`**`()` Returns the wages for this employee (i.e. a paycheck)

UML Class Diagram:

SalaryEmployee
- **salary**: *double*
+ **constructor** (name: *String*, salary: *double*) + **getPayment**()*: double* + **toString***(): String*

*Italicized class/method names within UML Class Diagram indicate abstract class/methods

SalaryEmployee Constructor Summary:

Constructor	Description
`SalaryEmployee(String name, double salary)`	Creates a SalaryEmployee instance

SalaryEmployee Method API:

Modifier and Type	Method and Description
`double`	**getPayment**() Returns paycheck amount which is salary / 12 months / 2x a month
`String`	**toString**() Returns employee text as "%s, salary:$%.02f" with name and salary

---

UML Class Diagram:

CommissionEmployee
- **commissionRate**: *double* - **totalSales**: *double*
+ **constructor** (name: *String*, rate: *double,* sales: *double*) + **getPayment**()*: double* + **toString***(): String*

*Italicized class/method names within UML Class Diagram indicate abstract class/methods

CommissionEmployee Constructor Summary:

Constructor	Description
`CommissionEmployee(String name, double rate, double sales)`	Creates a CommissionEmployee

CommissionEmployee Method API:

Modifier and Type	Method and Description
`double`	**getPayment**() Returns paycheck amount which is commission rate * total sale
`String`	**toString**() Returns employee text as "%s, commission:%.02f% @ $%.02f sales" with name, rate, sales

UML Class Diagram:

HourlyEmployee
- **hourlyRate**: *double* - **hoursWorked**: *double*
+ **constructor** (name: *String*, rate: *double,* hours: *double*) + **getPayment**()*: double* + **toString***(): String*

*Italicized class/method names within UML Class Diagram indicate abstract class/methods

HourlyEmployee Constructor Summary:

Constructor	Description
`HourlyEmployee(String name, double rate, double hours)`	Creates an HourlyEmployee

HourlyEmployee Method API:

Modifier and Type	Method and Description
`double`	**getPayment**() Returns paycheck amount which is hourly rate * hours worked
`String`	**toString**() Returns employee text as "%s, hourly:$%.02f% @ %.02f hours" with name, rate, hours

---

Tester Files:

Use the *TestEmployee.java* file to test your implementation. Compare your results with the *TestEmployee.txt* file.

Sample Method Calls	Sample Method Results

<pre>Employee[] e = new Employee[3];
 	
e[0] = new SalaryEmployee("Meg Manager",
                           50_000);
 	
e[1] = new CommissionEmployee("Sal Salesman",
                              .15, 3400);
 	
e[2] = new HourlyEmployee("Timmy Temp",
                              10.50, 25);
 	
for (Employee worker : e){
   System.out.println(worker);
   System.out.printf("Paycheck: $%.02f\n",
                       worker.getPayment());
}</pre>
	"Meg Manager, salary:$50000.00\n"
	"Paycheck: $2083.33\n"
	"Sal Salesman, commission:0.15% @ $3400.00 sales\n"
	"Paycheck: $510.00\n"
	"Timmy Temp, hourly:$10.50 @ 25.00 hours\n"
	"Paycheck: $262.50\n";

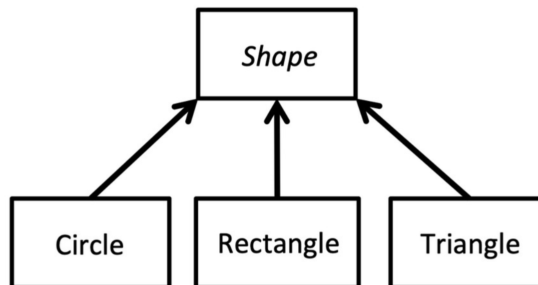
Problem 3: Shape (10 points)

(Software Design) Create an abstract class that represents a Shape and contains abstract method: area and perimeter. Create concrete classes: Circle, Rectangle, Triangle which extend the Shape class and implements the abstract methods. A circle has a radius, a rectangle has width and height, and a triangle has three sides.

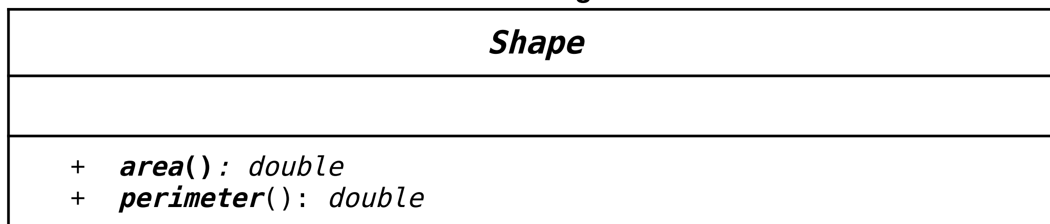
Software Architecture:

The **`Shape`** class is the abstract super class and must be instantiated by one of its concrete subclasses: Circle, Rectangle, or Triangle, which extends it.

UML Object Diagram



UML Class Diagram:



*Italicized class/method names within UML Class Diagram indicate abstract class/methods

Shape Method API:

Modifier and Type	Method and Description
`abstract double`	**`area`**`()` Returns the area of this shape
`abstract double`	**`perimeter`**`()` Returns the perimeter of this shape

UML Class Diagram:

Circle
- **radius**: *double*
+ **constructor** (radius: *double*) + **constructor** () + **area**()*: double* + **perimeter***(): double* + **toString***(): String*

*Italicized class/method names within UML Class Diagram indicate abstract class/methods

Circle Constructor Summary:

Constructor	Description
`Circle(double radius)`	Creates a Circle with given radius
`Circle()`	Creates a Circle with radius = 1

SalaryEmployee Method API:

Modifier and Type	Method and Description
double	**area**() Returns Math.PI * (radius)^2
double	**perimeter**() Returns 2 * Math.PI * radius
String	**toString**() Returns the String "Circle"

UML Class Diagram:

Rectangle
- **width**: *double* - **length**: *double*
+ **constructor** (width: *double*, length: *double*) + **constructor** () + **area**()*: double* + **perimeter***(): double* + **toString***(): String*

*Italicized class/method names within UML Class Diagram indicate abstract class/methods

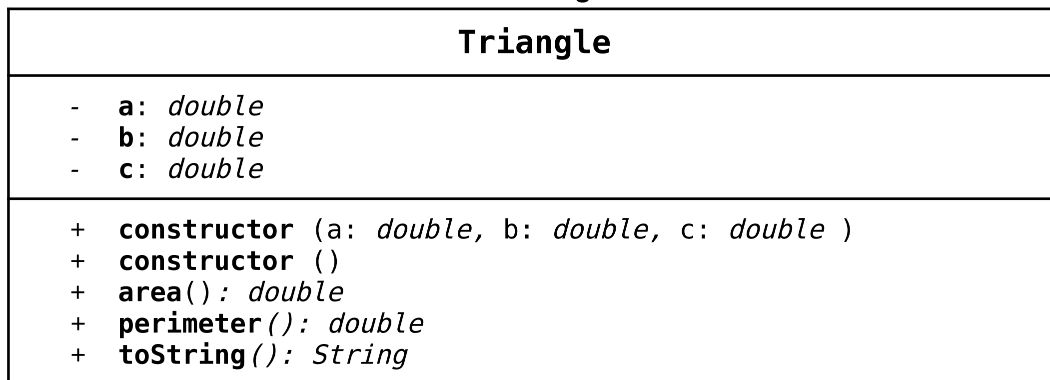
Rectangle Constructor Summary:

Constructor	Description
`Rectangle(double width, double length)`	Creates a Rectangle with given length, width
`Rectangle()`	Creates a Rectangle with width = 1, length = 1

Rectangle Method API:

Modifier and Type	Method and Description
double	**area**() Returns width * length
double	**perimeter**() Returns 2 * (width + length)
String	**toString**() Returns the String "Rectangle"

UML Class Diagram:



*Italicized class/method names within UML Class Diagram indicate abstract class/methods

Triangle Constructor Summary:

Constructor	Description
Triangle(double a, double b, double c)	Creates a Triangle with given sides a, b, c
Triangle()	Creates a Triangle with all sides = 1

Triangle Method API:

Modifier and Type	Method and Description
double	**area**() Returns area of triangle using heron's formula
double	**perimeter**() Returns a + b + c
String	**toString**() Returns the String "Triangle"

Tester Files:

Use the *TestShape.java* file to test your implementation. Compare your results with the *TestShape.txt* file.

<table>
<tr><th>Sample Method Calls</th><th>Sample Method Results</th></tr>
<tr><td><pre>
Shape[] shapes = new Shape[3];
shapes[0] = new Circle(3);
shapes[1] = new Rectangle(4,2);
shapes[2] = new Triangle(1,2,3);

for (Shape polygon : shapes){
   double area = polygon.area();
   double perimeter = polygon.perimeter();
   System.out.printf("%s: area: %.01f perimeter: %.01f\n",
                     polygon, area, perimeter);
}
</pre></td><td>"Circle: area: 28.3, perimeter: 18.8\n"
"Rectangle: area: 8.0, perimeter: 12.0\n"
"Triangle: area: 0.0, perimeter: 6.0\n"</td></tr>
</table>