Index

[Exercise 1: Subclassing and Inheritance 1 3](#_Toc65830565)

[Task 1 - Write code inside Circle.java file 3](#_Toc65830566)

[Task 2 - Created Cylinder class extended Circle class 4](#_Toc65830567)

[Task 3 - Write code inside TestCylinder.java file 4](#_Toc65830568)

[Task 4 - Execute your program 5](#_Toc65830569)

[Exercise 2: Subclassing and Inheritance 2 5](#_Toc65830570)

[Task 1 - Write code in Employee.java 6](#_Toc65830571)

[Task 2 - Create SalaryEmployee Class 7](#_Toc65830572)

[Task 3 - Write code in SalaryEmployee.java 7](#_Toc65830573)

[Task 4 - Create WageEmployee Class 7](#_Toc65830574)

[Task 5 - Write code in WageEmployee.java 8](#_Toc65830575)

[Task 6 - Write a Program class to Show Employee’s salary 8](#_Toc65830576)

[Task 7 - Execute your program 9](#_Toc65830577)

[Exercise 3: Polymorphism 9](#_Toc65830578)

[Task 1 - Create Shape Class 9](#_Toc65830579)

[Task 2 - Create Rectangle Class 10](#_Toc65830580)

[Task 3 - Write code in Rectangle.java 10](#_Toc65830581)

[Task 4 - Create Triangle Class 11](#_Toc65830582)

[Task 5 - Write code in Triangle.java 11](#_Toc65830583)

[Task 6 - Write a TestShape class 11](#_Toc65830584)

[Task 7 - Execute your program and see the result 12](#_Toc65830585)

[Exercise 4: Abstract 12](#_Toc65830586)

[Task 1 - Create Shape class (abstract class) 13](#_Toc65830587)

[Task 2 - Create Circle class 13](#_Toc65830588)

[Task 3 - Create Rectangle class 14](#_Toc65830589)

[Task 4 - Create Square class 15](#_Toc65830590)

[Task 5 - Test program 15](#_Toc65830591)

[Exercise 5: Interface 16](#_Toc65830592)

[Task 1 - Create Movable Interface 17](#_Toc65830593)

[Task 2 - Create MovablePoint Class implements Movable Interface 17](#_Toc65830594)

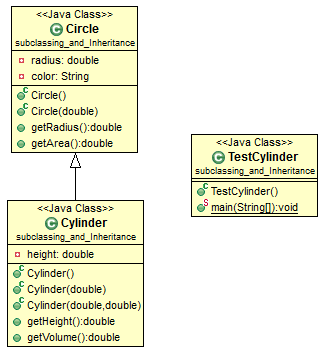
[Task 3 - Write code in MoveablePoint.java 18](#_Toc65830595)

[Task 4 - Create TestMovable Class to test 19](#_Toc65830596)

[Task 5 - Execute your program 19](#_Toc65830597)

1. Subclassing and Inheritance 1

In this exercise you will write a program to test how to use class and subclass. In this exercise you will create a base named Circle and another class named Cylinder Inherited Circle class, after that, write a small program to test these class. The diagram of your proram will be:



## Write code inside Circle.java file

* 1. **public** **class** Circle {
  2. **private** **double** radius;
  3. **private** String color;
  5. // 1st constructor, which sets both radius and color to default
  6. **public** Circle() {
  7. radius = 1.0;
  8. color = "red";
  9. }
  11. // 2nd constructor with given radius, but color default
  12. **public** Circle(**double** r) {
  13. radius = r;
  14. color = "red";
  15. }
  17. // A public method for retrieving the radius
  18. **public** **double** getRadius() {
  19. **return** radius;
  20. }
  22. // A public method for computing the area of circle
  23. **public** **double** getArea() {
  24. **return** radius\*radius\*Math.*PI*;
  25. }
  26. }

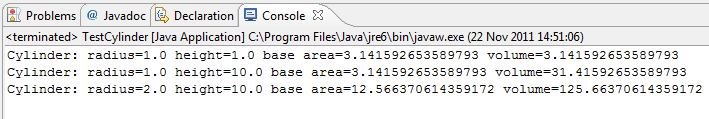
## Created Cylinder class extended Circle class

* 1. **public** **class** Cylinder **extends** Circle {
  2. **private** **double** height;
  4. // Constructor with default color, radius and height
  5. **public** Cylinder() {
  6. **super**(); // call superclass no-arg constructor Circle()
  7. height = 1.0;
  8. }
  9. // Constructor with default radius, color but given height
  10. **public** Cylinder(**double** height) {
  11. **super**(); // call superclass no-arg constructor Circle()
  12. **this**.height = height;
  13. }
  14. // Constructor with default color, but given radius, height
  15. **public** Cylinder(**double** radius, **double** height) {
  16. **super**(radius); // call superclass constructor Circle(r)
  17. **this**.height = height;
  18. }
  20. // A public method for retrieving the height
  21. **public** **double** getHeight() {
  22. **return** height;
  23. }
  25. // A public method for computing the volume of cylinder
  26. // use superclass method getArea() to get the base area
  27. **public** **double** getVolume() {
  28. **return** getArea()\*height;
  29. }
  30. }

## Write code inside TestCylinder.java file

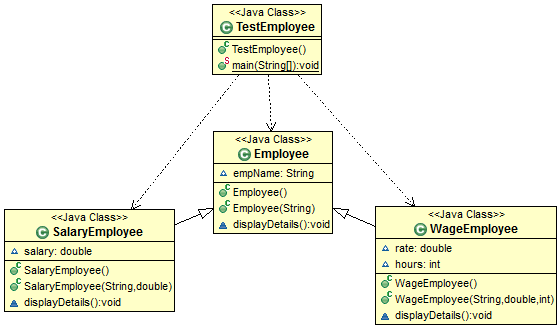
* 1. **public** **class** TestCylinder { // save as "TestCylinder.java"
  2. **public** **static** **void** main (String[] args) {
  3. // Declare and allocate a new instance of cylinder
  4. // with default color, radius, and height
  5. Cylinder c1 = **new** Cylinder();
  6. System.*out*.println("Cylinder:"
  7. + " radius=" + c1.getRadius()
  8. + " height=" + c1.getHeight()
  9. + " base area=" + c1.getArea()
  10. + " volume=" + c1.getVolume());
  12. // Declare and allocate a new instance of cylinder
  13. // specifying height, with default color and radius
  14. Cylinder c2 = **new** Cylinder(10.0);
  15. System.*out*.println("Cylinder:"
  16. + " radius=" + c2.getRadius()
  17. + " height=" + c2.getHeight()
  18. + " base area=" + c2.getArea()
  19. + " volume=" + c2.getVolume());
  21. // Declare and allocate a new instance of cylinder
  22. // specifying radius and height, with default color
  23. Cylinder c3 = **new** Cylinder(2.0, 10.0);
  24. System.*out*.println("Cylinder:"
  25. + " radius=" + c3.getRadius()
  26. + " height=" + c3.getHeight()
  27. + " base area=" + c3.getArea()
  28. + " volume=" + c3.getVolume());
  29. }
  30. }

## Execute your program

* 1. 

1. Subclassing and Inheritance 2

Class diagram of the program:



## Write code in Employee.java

### Create 2 Constructors

* 1. **public** Employee() {
  2. }
  4. **public** Employee(String name) {
  5. empName = name;
  6. }

### Instance variable to store the employee's name.

* 1. String empName;

### Method to display the name.

* 1. **void** displayDetails() {
  2. System.*out*.printf("Employee Name: %s", empName);
  3. }

## Create SalaryEmployee Class

* 1. **public** **class** SalaryEmployee **extends** Employee {
  2. }

## Write code in SalaryEmployee.java

### Create 2 Constructors

* 1. **public** SalaryEmployee() {
  2. }
  3. **public** SalaryEmployee(String name, **double** currentSalary) {
  4. **super**(name);
  5. salary = currentSalary;
  6. }

### Instance variable salary to store salary for salaried employees.

* 1. **double** salary;

### Method to display the details of salaried employees.

* 1. **void** displayDetails() {
  2. **super**.displayDetails();
  3. System.*out*.printf("\n" + empName + "'s Salary: %.2f\n", salary);
  4. }

## Create WageEmployee Class

* 1. **public** **class** WageEmployee **extends** Employee {
  2. }

## Write code in WageEmployee.java

### Create two constructors

* 1. **public** WageEmployee() {
  2. }
  4. **public** WageEmployee(String name, **double** wageRate, **int** wageHours) {
  5. **super**(name);
  6. rate = wageRate;
  7. hours = wageHours;
  8. }

### Variables to store rate for one hour and working hours

* 1. **double** rate;
  2. **int** hours;

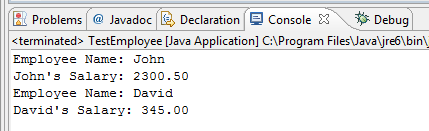
### Method to display the details of waged employees.

* 1. **void** displayDetails() {
  2. **super**.displayDetails();
  3. System.*out*.printf("\n" + empName + "'s Salary: %.2f\n", rate \* hours);
  4. }

## Write a Program class to Show Employee’s salary

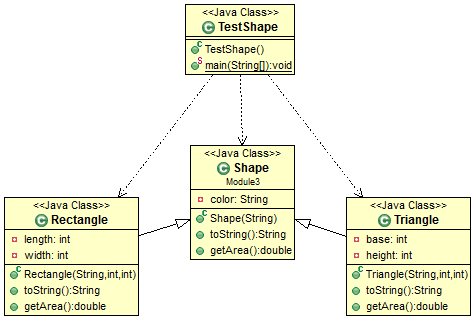
* 1. **public** **static** **void** main(String[] args) {
  3. Employee objJohn = **new** SalaryEmployee("John", 2300.50);
  4. objJohn.displayDetails();
  6. Employee objDavid = **new** WageEmployee("David", 34.50, 10);
  7. objDavid.displayDetails();
  8. }

## Execute your program

* 1. 

1. Polymorphism

In this exercise, you will define many kinds of shapes, such as triangle, rectangle and so on. You will design a superclass called Shape, which defines the public interface (or behaviours) of all the shapes. The diagram class are shown below:



## Create Shape Class

* 1. // Instance variable
  2. **private** String color;
  4. // Constructor
  5. **public** Shape (String color) {
  6. **this**.color = color;
  7. }
  9. **public** String toString() {
  10. **return** "Shape of color=\"" + color + "\"";
  11. }
  13. // All shapes must has a method called getArea()
  14. **public** **double** getArea() {
  15. System.*err*.println("Shape unknown! Cannot compute area!");
  16. **return** 0; // Need a return to compile the program
  17. }

## Create Rectangle Class

* 1. **public** **class** Rectangle **extends** Shape {
  2. }

## Write code in Rectangle.java

### Instance variables

* 1. **private** **int** length;
  2. **private** **int** width;

### Constructor

* 1. **public** Rectangle(String color, **int** length, **int** width) {
  2. **super**(color);
  3. **this**.length = length;
  4. **this**.width = width;
  5. }

### Two override methods

* 1. @Override
  2. **public** String toString() {
  3. **return** "Rectangle of length=" + length + " and width=" + width + ", subclass of " + **super**.toString();
  4. }
  6. @Override
  7. **public** **double** getArea() {
  8. **return** length\*width;
  9. }

## Create Triangle Class

* 1. **public class** Triangle **extends** Shape {
  2. }

## Write code in Triangle.java

### Instance variables

* 1. **private** **int** base;
  2. **private** **int** height;

### Constructor

* 1. **public** Triangle(String color, **int** base, **int** height) {
  2. **super**(color);
  3. **this**.base = base;
  4. **this**.height = height;
  5. }

### Two override methods

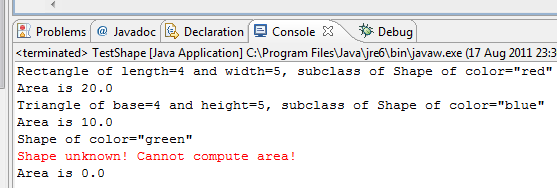
* 1. @Override
  2. **public** String toString() {
  3. **return** "Triangle of base=" + base + " and height=" + height + ", subclass of " + **super**.toString();
  4. }
  6. @Override
  7. **public** **double** getArea() {
  8. **return** 0.5\*base\*height;
  9. }

## Write a TestShape class

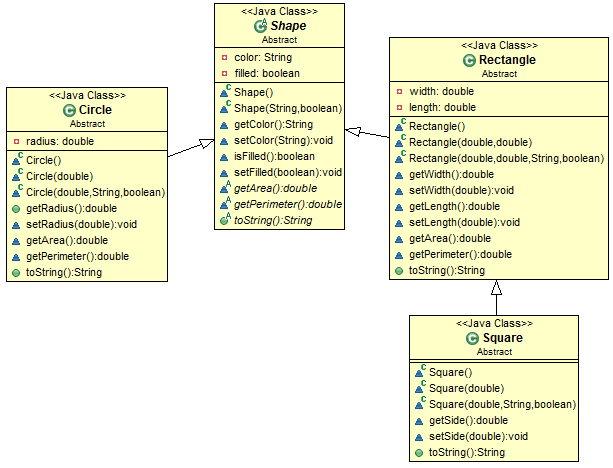
### Write code for main() method:

* 1. **public** **static** **void** main(String[] args) {
  2. Shape s1 = **new** Rectangle("red", 4, 5);
  3. System.*out*.println(s1);
  4. System.*out*.println("Area is " + s1.getArea());
  6. Shape s2 = **new** Triangle("blue", 4, 5);
  7. System.*out*.println(s2);
  8. System.*out*.println("Area is " + s2.getArea());
  10. Shape s3 = **new** Shape("green");
  11. System.*out*.println(s3);
  12. System.*out*.println("Area is " + s3.getArea());
  13. }

## Execute your program and see the result



1. Abstract



## Create Shape class (abstract class)

1. public abstract class Shape {
2. private String color;
3. private boolean filled;
4. Shape()
5. {
7. }
8. Shape(String color,boolean filled)
9. {
10. …
11. }
12. String getColor()
13. {
14. …
15. }
16. void setColor(String color)
17. {
18. …
19. }
20. boolean isFilled()
21. {
22. …
23. }
24. void setFilled(boolean filled)
25. {
26. …
27. }
28. abstract double getArea();
29. abstract double getPerimeter();
30. public abstract String toString();
31. }

## Create Circle class

1. public class Circle extends Shape {
2. private double radius;
3. Circle()
4. {
5. }
6. Circle(double radius)
7. {
8. …
9. }
10. Circle(double radius,String color,boolean filled)
11. {
12. …
13. }
14. public double getRadius()
15. {
16. …
17. }
18. void setRadius(double radius)
19. {
20. …
21. }
22. double getArea()
23. {
24. …
25. }
26. double getPerimeter()
27. {
28. …
29. }
30. public String toString()
31. {
32. …
33. }
34. }

## Create Rectangle class

* 1. **public** **class** Rectangle **extends** Shape {
  2. **private** **double** width;
  3. **private** **double** length;
  4. Rectangle()
  5. {
  6. }
  7. Rectangle(**double** width,**double** length)
  8. {
  9. **…**
  10. }
  11. Rectangle(**double** width,**double** length,String color,**boolean** filled)
  12. {
  13. **…**
  14. }
  15. **double** getWidth()
  16. {
  17. **…**
  18. }
  19. **void** setWidth(**double** width)
  20. {
  21. **…**
  22. }
  23. **double** getLength()
  24. {
  25. **…**
  26. }
  27. **void** setLength(**double** length)
  28. {
  29. **…**
  30. }
  32. **double** getArea()
  33. {
  34. **…**
  35. }
  36. **double** getPerimeter()
  37. {
  38. **…**
  39. }
  40. **public** String toString()
  41. {
  42. …
  43. }
  44. }

## Create Square class

* 1. **public** **class** Square **extends** Rectangle {
  2. …
  3. }

## Test program

Using below code to test all class created in above code:

In these code there are some error, explain, fix it and run the program.

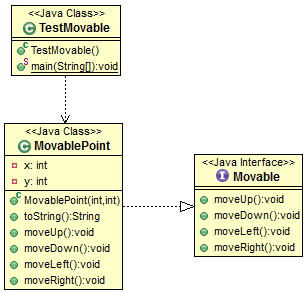
* 1. **public** **class** TestAllShape {
  2. **public** **static** **void** main(String[] args) {
  3. Shape s1 = **new** Circle(5.5, "RED", **false**);
  4. System.*out*.println(s1);
  5. System.*out*.println(s1.getArea());
  6. System.*out*.println(s1.getPerimeter());
  7. System.*out*.println(s1.getColor());
  8. System.*out*.println(s1.isFilled());
  9. System.*out*.println(s1.getRadius());
  11. Circle c1 = (Circle)s1;
  12. System.*out*.println(c1);
  13. System.*out*.println(c1.getArea());
  14. System.*out*.println(c1.getPerimeter());
  15. System.*out*.println(c1.getColor());
  16. System.*out*.println(c1.isFilled());
  17. System.*out*.println(c1.getRadius());
  19. Shape s2 = **new** Shape();
  21. Shape s3 = **new** Rectangle(1.0, 2.0, "RED", **false**); // Upcast
  22. System.*out*.println(s3);
  23. System.*out*.println(s3.getArea());
  24. System.*out*.println(s3.getPerimeter());
  25. System.*out*.println(s3.getColor());
  26. System.*out*.println(s3.getLength());
  28. Rectangle r1 = (Rectangle)s3; // downcast
  29. System.*out*.println(r1);
  30. System.*out*.println(r1.getArea());
  31. System.*out*.println(r1.getColor());
  32. System.*out*.println(r1.getLength());
  34. Shape s4 = **new** Square(6.6); // Upcast
  35. System.*out*.println(s4);
  36. System.*out*.println(s4.getArea());
  37. System.*out*.println(s4.getColor());
  38. System.*out*.println(s4.getSide());
  40. Rectangle r2 = (Rectangle)s4;
  41. System.*out*.println(r2);
  42. System.*out*.println(r2.getArea());
  43. System.*out*.println(r2.getColor());
  44. System.*out*.println(r2.getSide());
  45. System.*out*.println(r2.getLength());

  48. Square sq1 = (Square)sq1;
  49. System.*out*.println(sq1);
  50. System.*out*.println(sq1.getArea());
  51. System.*out*.println(sq1.getColor());
  52. System.*out*.println(sq1.getSide());
  53. System.*out*.println(sq1.getLength());
  54. }
  55. }

1. Interface

In Exercise, suppose that the application involves many objects that can move. You could define an interface called movable, containing the signatures of the various movement methods.

Class diagram of the program are shown below:



## Create Movable Interface

* 1. **public interface** Moveable {
  2. }

### Write code in Moveable.java

* 1. **public interface** Movable {
  2. **public void** moveUp();
  3. **public void** moveDown();
  4. **public void** moveLeft();
  5. **public void** moveRight();
  6. }

## Create MovablePoint Class implements Movable Interface

### See the code what IDE generate:

* 1. **public class** MoveablePoint **implements** Movable {
  2. @Override
  3. **public void** moveUp() {
  4. // TODO Auto-generated method stub
  5. }
  6. @Override
  7. **public void** moveDown() {
  8. // TODO Auto-generated method stub
  9. }
  10. @Override
  11. **public void** moveLeft() {
  12. // TODO Auto-generated method stub
  13. }
  14. @Override
  15. **public void** moveRight() {
  16. // TODO Auto-generated method stub
  17. }
  18. }

## Write code in MoveablePoint.java

### Instance variables - (x, y) coordinates of the point

* 1. **private** **int** x;
  2. **private** **int** y;

### Constructor

* 1. **public** MovablePoint(**int** x, **int** y) {
  2. **this**.x = x;
  3. **this**.y = y;
  4. }
  6. **public** String toString() {
  7. **return** "Point at (" + x + "," + y + ")";
  8. }

### Implement abstract methods defined in the interface Movable

* 1. @Override
  2. **public** **void** moveUp() {
  3. y--;
  4. }
  6. @Override
  7. **public** **void** moveDown() {
  8. y++;
  9. }
  11. @Override
  12. **public** **void** moveLeft() {
  13. x--;
  14. }
  16. @Override
  17. **public** **void** moveRight() {
  18. x++;
  19. }

## Create TestMovable Class to test

* 1. **public** **static** **void** main(String[] args) {
  2. Movable m1 = **new** MovablePoint(5, 5);
  3. System.*out*.println(m1);
  4. m1.moveDown();
  5. System.*out*.println(m1);
  6. m1.moveRight();
  7. System.*out*.println(m1);
  8. }

## Execute your program

