JAVA PROGRAMMING

Tutorial 06

Activity 1: Shape2D

The Point2D class represents a point in a two-dimensional space, with x and y coordinates. It serves as a fundamental component for defining shapes' positions.

The Shape2D abstract class encapsulates common attributes and behaviors of geometric shapes. It includes fields for color, filled status, and position (as a Point2D). It implements an interface with methods to calculate area and perimeter. Shape2D serves as a blueprint for specific types of 2D shapes such as Circle, Rectangle, etc.

In this exercise, your task is to implement the relevant classes shown in the following UML diagram. Note that you should mark all overridden methods with the annotation `@Override`.

Notice:

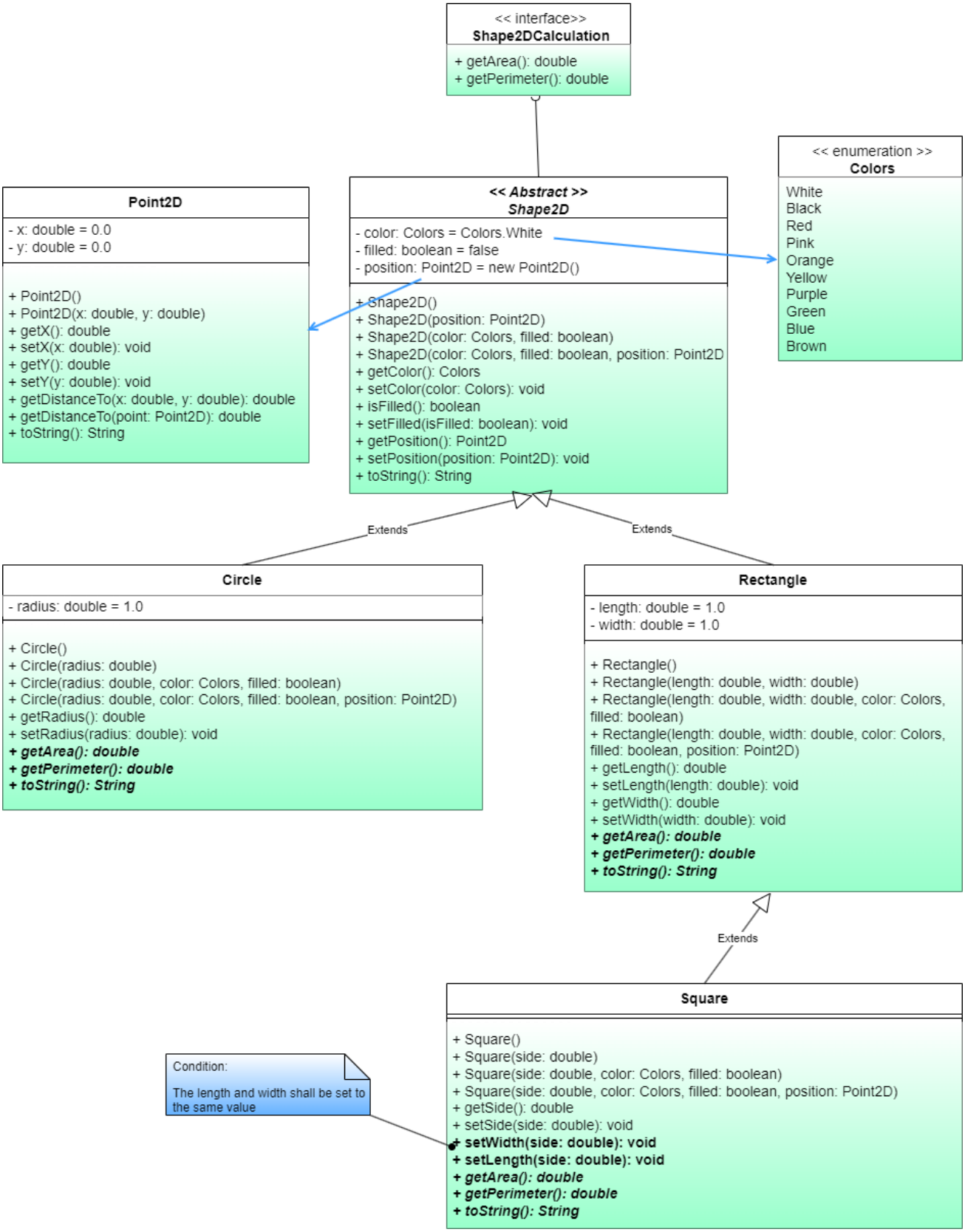
The super keyword refers to superclass (parent) objects. It is used to call superclass methods, and to access the superclass constructor. The most common use of the super keyword is to eliminate the confusion between super classes and subclasses that have methods with the same name.

The syntax to call a superclass's constructor is:

super(); or **super**(parameters);

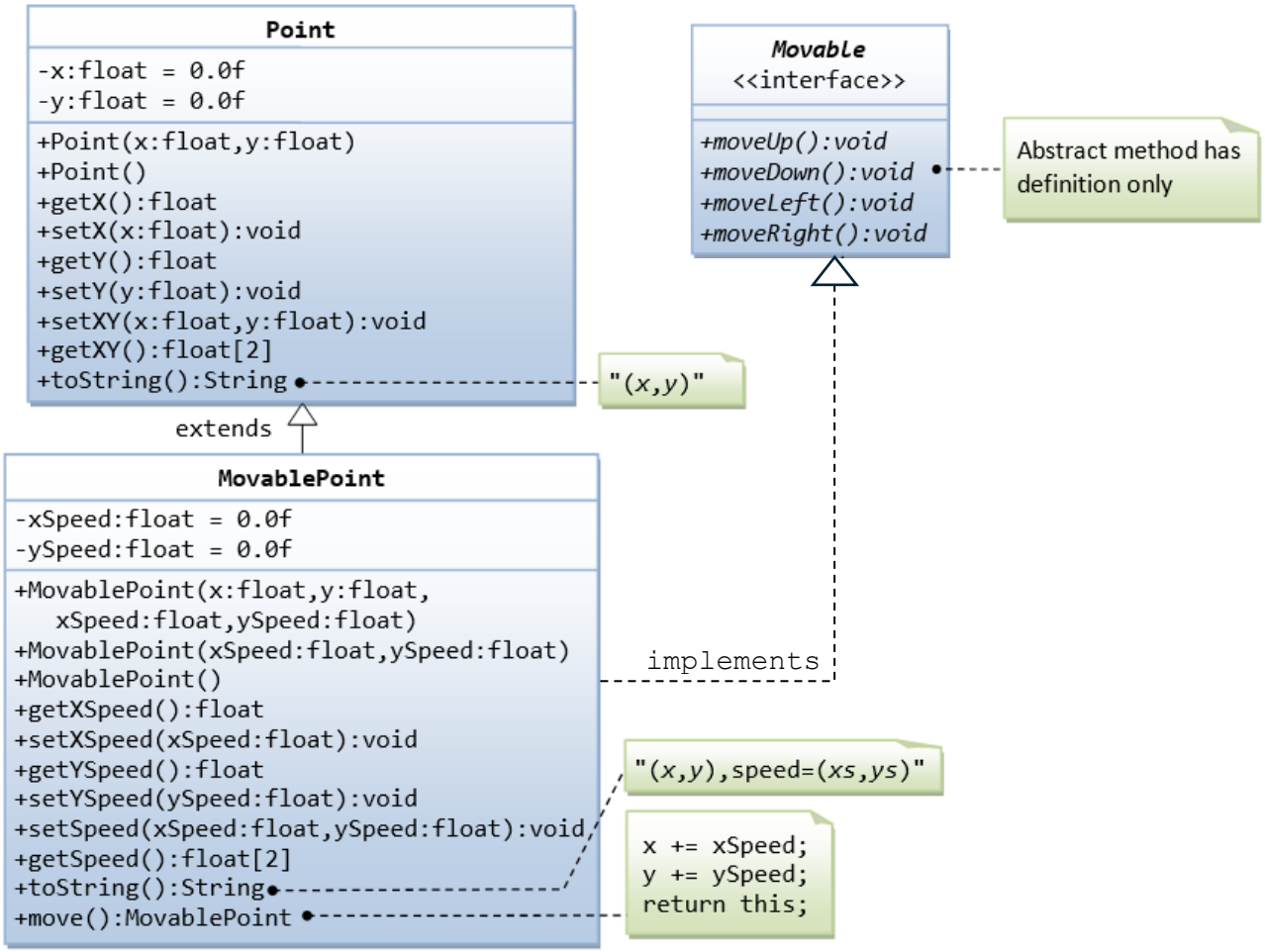
The syntax to call a superclass's method is:

super.method(); or **super**.method(parameters);


<< interface>>
Shape2DCalculation
+ getArea(): double
+ getPerimeter(): double
<< enumeration >>
Colors
White
Black
Red
Pink
Orange
Yellow
Purple
Green
Blue
Brown
Point2D
- x: double = 0.0
- y: double = 0.0
+ Point2D()
+ Point2D(x: double, y: double)
+ getX(): double
+ setX(x: double): void
+ getY(): double
+ setY(y: double): void
+ getDistanceTo(x: double, y: double): double
+ getDistanceTo(point: Point2D): double
+ toString(): String
<< Abstract >>
Shape2D
- color: Colors = Colors.White
- filled: boolean = false
- position: Point2D = new Point2D()
+ Shape2D()
+ Shape2D(position: Point2D)
+ Shape2D(color: Colors, filled: boolean)
+ Shape2D(color: Colors, filled: boolean, position: Point2D
+ getColor(): Colors
+ setColor(color: Colors): void
+ isFilled(): boolean
+ setFilled(isFilled: boolean): void
+ getPosition(): Point2D
+ setPosition(position: Point2D): void
+ toString(): String
Extends
Extends
Circle
- radius: double = 1.0
+ Circle()
+ Circle(radius: double)
+ Circle(radius: double, color: Colors, filled: boolean)
+ Circle(radius: double, color: Colors, filled: boolean, position: Point2D)
+ getRadius(): double
+ setRadius(radius: double): void
+ getArea(): double
+ getPerimeter(): double
+ toString(): String
Rectangle
- length: double = 1.0
- width: double = 1.0
+ Rectangle()
+ Rectangle(length: double, width: double)
+ Rectangle(length: double, width: double, color: Colors, filled: boolean)
+ Rectangle(length: double, width: double, color: Colors, filled: boolean, position: Point2D)
+ getLength(): double
+ setLength(length: double): void
+ getWidth(): double
+ setWidth(width: double): void
+ getArea(): double
+ getPerimeter(): double
+ toString(): String
Extends
Square
+ Square()
+ Square(side: double)
+ Square(side: double, color: Colors, filled: boolean)
+ Square(side: double, color: Colors, filled: boolean, position: Point2D)
+ getSide(): double
+ setSide(side: double): void
+ setWidth(side: double): void
+ setLength(side: double): void
+ getArea(): double
+ getPerimeter(): double
+ toString(): String
Condition:
The length and width shall be set to the same value

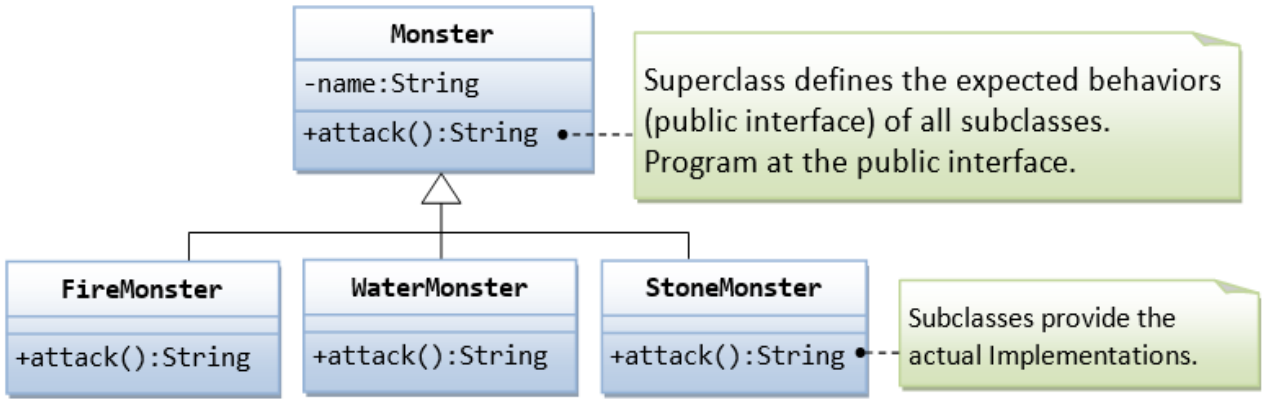
Activity 2: MovablePoint

Suppose the application requires managing multiple moving objects, a refined approach would involve defining an interface named Movable, encapsulating essential methods for movement. This interface would serve as a contract, ensuring consistent behavior across different movable objects. Additionally, to ensure the correctness and functionality of these classes, a comprehensive test suite should be developed, covering various scenarios and test cases.



Activity 3: Monster

In this exercise, you will design a superclass called Monster and define the method attack() in the superclass. The subclasses shall then provides their actual implementation. In the main program, we declare instances of superclass, substituted with actual subclass; and invoke method defined in the superclass. The UML diagram:



The main program (MonsterTest.java) shall be implemented following the specifications defined in the superclass and its subclasses. You are required to declare instances of the superclass, substituted by instances of the subclasses, and invoke the method defined in the superclass.

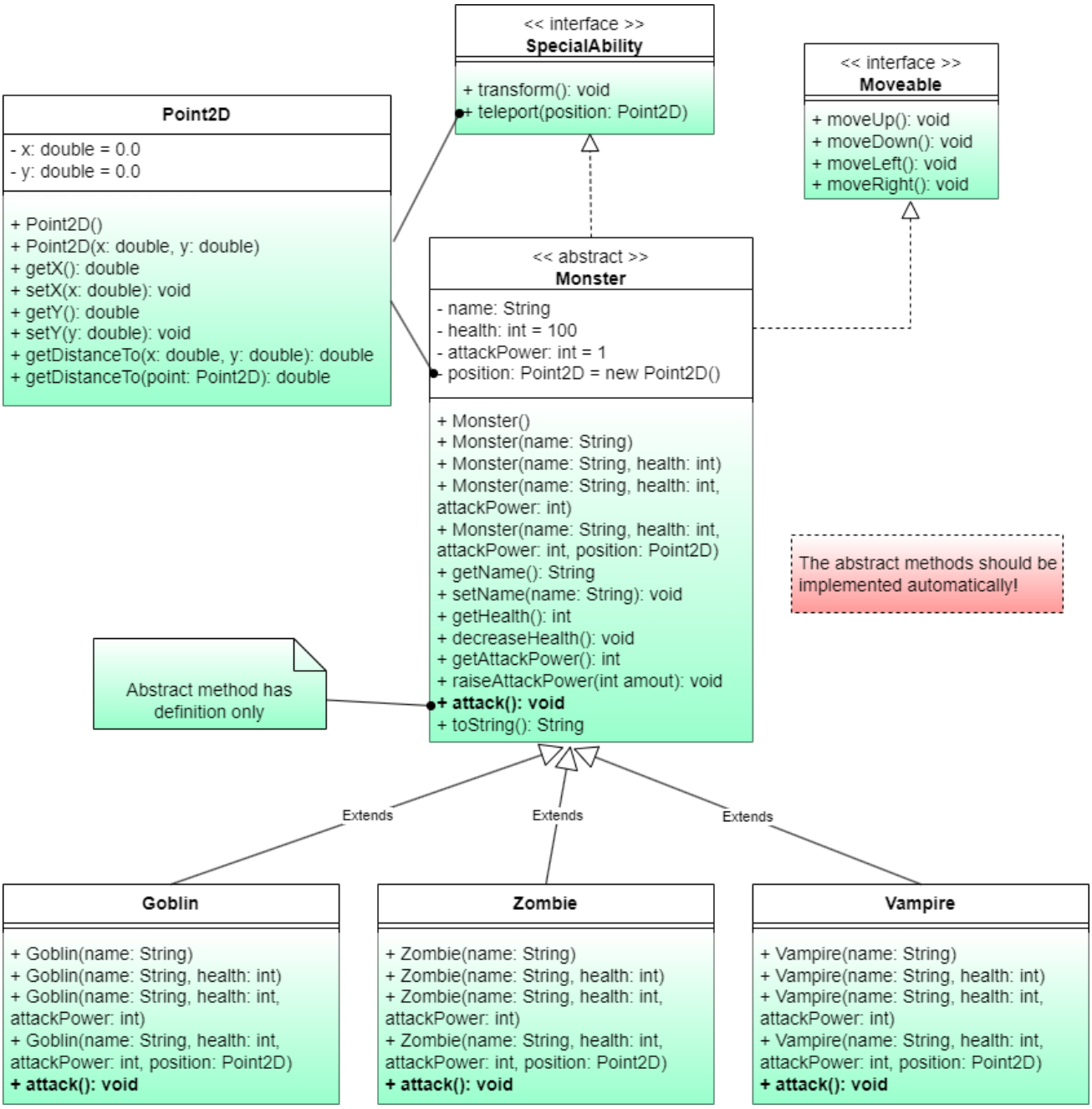
Ensure the following functionalities in your implementation:

- Declare instances of the superclass (Monster), substituting them with instances of the subclasses (FireMonster, WaterMonster, StoneMonster).
- Demonstrate polymorphic behavior by invoking the attack() method on these instances, ensuring that the respective subclass's implementation is executed.
- Reassign one of the superclass instances with a new instance of a subclass and verify that the appropriate attack method is called.

- Additionally, handle the scenario where an instance of the superclass is directly created, and ensure an appropriate message is displayed for this case.

Activity 4: Monster (Enhanced - Optional)

In this exercise, your mission is to refine the existing program by implementing the specified classes outlined in the provided diagram. Implement the classes defined as shown in the diagram.



Submission

Submit a zip file containing all Java programs to this tutorial's submission box in the course website on FIT Portal.