Question 4: Inheritance and Polymorphism in Java and Python

IST603 - Object-Oriented Programming with Java and Python

Total Marks: 26 marks

Part a) Python Section

i) Class Shape

Requirements:

- Write a class called `Shape`
- The initializer stores the string `'shape'` in an instance variable called `shape`
- Implement `__str__` method that returns: `"I am a shape"` (where `shape` is replaced by the instance variable)

Solution:

```
class Shape:
    """Base class for all shapes."""

    def __init__(self):
        """Initializer that stores 'shape' in instance variable."""
        self.shape = 'shape'

    def __str__(self):
        """Returns string representation of the shape."""
        return f"I am a {self.shape}"
```

Explanation:

- `__init__` initializes `self.shape = 'shape'`
- `__str__` is a special method that defines string representation
- Returns formatted string using f-string

---

ii) Class Polygon

Requirements:

- Create `Polygon` as a subclass of `Shape`
- `ii-1)` Initializer assigns `'polygon'` to `shape` and `None` to list `side_lengths`
- `ii-2)` Add method `compute_perimeter` that returns sum of values in `side_lengths` using `sum()`
- `ii-3)` Add method `get_number_of_edges` that returns length of `side_lengths` using `len()`

Solution:

```
class Polygon(Shape):
    """Class representing a polygon, subclass of Shape."""

    def __init__(self):
        """Initializer for Polygon."""
        super().__init__()  # Call parent initializer
        self.shape = 'polygon'
        self.side_lengths = None

    def compute_perimeter(self):
        """
        Returns the sum of values in side_lengths.

        Returns:
        - Sum of side lengths, or 0 if side_lengths is None
        """
        if self.side_lengths is None:
            return 0
        return sum(self.side_lengths)

    def get_number_of_edges(self):
        """
        Returns the length of the side_lengths list.

        Returns:
        - Number of edges, or 0 if side_lengths is None
        """
        if self.side_lengths is None:
            return 0
        return len(self.side_lengths)
```

Explanation:

- `Polygon(Shape)` indicates inheritance from `Shape`
- `super().__init__()` calls parent initializer
- Overrides `shape` to `'polygon'`
- `side_lengths` initialized to `None`
- `compute_perimeter()` uses built-in `sum()` function
- `get_number_of_edges()` uses built-in `len()` function
- Both methods handle `None` case

---

iii) Class Rectangle

Requirements:

- Create `Rectangle` as a subclass of `Polygon`
- Initializer assigns `'rectangle'` to `shape` and `[1, 1, 1, 1]` to `side_lengths`

Solution:

```
class Rectangle(Polygon):
    """Class representing a rectangle, subclass of Polygon."""

    def __init__(self):
        """Initializer for Rectangle."""
        super().__init__()  # Call parent initializer
        self.shape = 'rectangle'
        self.side_lengths = [1, 1, 1, 1]
```

Explanation:

- `Rectangle(Polygon)` inherits from `Polygon`
- Calls `super().__init__()` to initialize parent
- Sets `shape = 'rectangle'`
- Initializes `side_lengths = [1, 1, 1, 1]` (4 sides of length 1)

---

iv) Class Triangle

Requirements:

- Create `Triangle` as a subclass of `Polygon`
- Initializer assigns `'triangle'` to `shape` and `[2, 2, 2]` to `side_lengths`

Solution:

```
class Triangle(Polygon):
    """Class representing a triangle, subclass of Polygon."""

    def __init__(self):
        """Initializer for Triangle."""
        super().__init__()  # Call parent initializer
        self.shape = 'triangle'
        self.side_lengths = [2, 2, 2]
```

Explanation:

- `Triangle(Polygon)` inherits from `Polygon`
- Calls `super().__init__()` to initialize parent
- Sets `shape = 'triangle'`
- Initializes `side_lengths = [2, 2, 2]` (3 sides of length 2)

---

v) Outputs

Given Code:

```
# v-1)
s = Shape()
print(s)

# v-2)
p = Polygon()
print(p)

# v-3)
rect = Rectangle()
print(rect)
rect.compute_perimeter()

# v-4)
tri = Triangle()
print(tri)
tri.compute_perimeter()
```

Output Analysis:

v-1) `s = Shape(); print(s)`**:**

- Creates `Shape` object
- `shape = 'shape'`
- `print(s)` calls `__str__()` which returns `"I am a shape"`

Output:

```
I am a shape
```

v-2) `p = Polygon(); print(p)`**:**

- Creates `Polygon` object
- `shape = 'polygon'`
- `print(p)` calls `__str__()` which returns `"I am a polygon"`

Output:

```
I am a polygon
```

v-3) `rect = Rectangle(); print(rect); rect.compute_perimeter()`**:**

- Creates `Rectangle` object
- `shape = 'rectangle'`, `side_lengths = [1, 1, 1, 1]`
- `print(rect)` calls `__str__()` which returns `"I am a rectangle"`
- `rect.compute_perimeter()` returns `sum([1, 1, 1, 1]) = 4`
- **Note:** The method returns 4 but doesn't print it (no print statement)

Output:

```
I am a rectangle
```

v-4) `tri = Triangle(); print(tri); tri.compute_perimeter()`**:**

- Creates `Triangle` object
- `shape = 'triangle'`, `side_lengths = [2, 2, 2]`
- `print(tri)` calls `__str__()` which returns `"I am a triangle"`
- `tri.compute_perimeter()` returns `sum([2, 2, 2]) = 6`
- **Note:** The method returns 6 but doesn't print it (no print statement)

Output:

```
I am a triangle
```

Complete Output:

```
I am a shape
I am a polygon
I am a rectangle
I am a triangle
```

Part b) Java Section

i) Java Code for Classes (a-i to a-iv)

Solution:

```java
// i) Class Shape
class Shape {
    private String shape;

    public Shape() {
        this.shape = "shape";
    }

    @Override
    public String toString() {
        return "I am a " + this.shape;
    }
}

// ii) Class Polygon
```

```
class Polygon extends Shape {
    private java.util.List<Integer> sideLengths;

    public Polygon() {
        super();
        this.shape = "polygon";
        this.sideLengths = null;
    }

    public int computePerimeter() {
        if (sideLengths == null) {
            return 0;
        }
        int sum = 0;
        for (int length : sideLengths) {
            sum += length;
        }
        return sum;
    }

    public int getNumberOfEdges() {
        if (sideLengths == null) {
            return 0;
        }
        return sideLengths.size();
    }
}

// iii) Class Rectangle
class Rectangle extends Polygon {
    public Rectangle() {
        super();
        this.shape = "rectangle";
        this.sideLengths = new java.util.ArrayList<>();
        sideLengths.add(1);
        sideLengths.add(1);
        sideLengths.add(1);
        sideLengths.add(1);
    }
}

// iv) Class Triangle
class Triangle extends Polygon {
    public Triangle() {
        super();
        this.shape = "triangle";
        this.sideLengths = new java.util.ArrayList<>();
        sideLengths.add(2);
        sideLengths.add(2);
        sideLengths.add(2);
    }
}
```

Note: The above code has an issue - `shape` is `private` in `Shape`, so subclasses can't access it. Here's the corrected version:

Corrected Solution:

```
// i) Class Shape
class Shape {
    protected String shape;  // Changed to protected for inheritance

    public Shape() {
        this.shape = "shape";
    }

    @Override
    public String toString() {
        return "I am a " + this.shape;
    }
}

// ii) Class Polygon
class Polygon extends Shape {
    protected java.util.List<Integer> sideLengths;  // protected for inheritance

    public Polygon() {
        super();
        this.shape = "polygon";
        this.sideLengths = null;
    }

    public int computePerimeter() {
        if (sideLengths == null) {
            return 0;
        }
        int sum = 0;
        for (int length : sideLengths) {
            sum += length;
        }
        return sum;
    }

    public int getNumberOfEdges() {
        if (sideLengths == null) {
            return 0;
        }
        return sideLengths.size();
    }
}

// iii) Class Rectangle
class Rectangle extends Polygon {
    public Rectangle() {
        super();
```

```
        this.shape = "rectangle";
        this.sideLengths = new java.util.ArrayList<>();
        sideLengths.add(1);
        sideLengths.add(1);
        sideLengths.add(1);
        sideLengths.add(1);
    }
}

// iv) Class Triangle
class Triangle extends Polygon {
    public Triangle() {
        super();
        this.shape = "triangle";
        this.sideLengths = new java.util.ArrayList<>();
        sideLengths.add(2);
        sideLengths.add(2);
        sideLengths.add(2);
    }
}
```

Key Differences from Python:

- Uses `extends` keyword for inheritance
- `protected` access modifier allows subclasses to access fields
- Uses `super()` to call parent constructor
- `toString()` method instead of `__str__()`
- Uses `ArrayList` for dynamic lists
- Explicit type declarations

---

ii) Java Statements for (a-v)

Solution:

```
// v-1)
Shape s = new Shape();
System.out.println(s);

// v-2)
Polygon p = new Polygon();
System.out.println(p);

// v-3)
Rectangle rect = new Rectangle();
System.out.println(rect);
rect.computePerimeter();

// v-4)
Triangle tri = new Triangle();
```

```
System.out.println(tri);
tri.computePerimeter();
```

Output:

```
I am a shape
I am a polygon
I am a rectangle
I am a triangle
```

Complete Java Test Program:

```
public class ShapeTest {
    public static void main(String[] args) {
        // v-1)
        Shape s = new Shape();
        System.out.println(s);

        // v-2)
        Polygon p = new Polygon();
        System.out.println(p);

        // v-3)
        Rectangle rect = new Rectangle();
        System.out.println(rect);
        int rectPerimeter = rect.computePerimeter();
        System.out.println("Rectangle perimeter: " + rectPerimeter);

        // v-4)
        Triangle tri = new Triangle();
        System.out.println(tri);
        int triPerimeter = tri.computePerimeter();
        System.out.println("Triangle perimeter: " + triPerimeter);
    }
}
```

Output:

```
I am a shape
I am a polygon
I am a rectangle
Rectangle perimeter: 4
I am a triangle
Triangle perimeter: 6
```

Complete Python Code

```
class Shape:
    def __init__(self):
        self.shape = 'shape'

    def __str__(self):
        return f"I am a {self.shape}"


class Polygon(Shape):
    def __init__(self):
        super().__init__()
        self.shape = 'polygon'
        self.side_lengths = None

    def compute_perimeter(self):
        if self.side_lengths is None:
            return 0
        return sum(self.side_lengths)

    def get_number_of_edges(self):
        if self.side_lengths is None:
            return 0
        return len(self.side_lengths)


class Rectangle(Polygon):
    def __init__(self):
        super().__init__()
        self.shape = 'rectangle'
        self.side_lengths = [1, 1, 1, 1]


class Triangle(Polygon):
    def __init__(self):
        super().__init__()
        self.shape = 'triangle'
        self.side_lengths = [2, 2, 2]


# Test code
if __name__ == "__main__":
    # v-1)
    s = Shape()
    print(s)

    # v-2)
    p = Polygon()
    print(p)

    # v-3)
```

```
rect = Rectangle()
print(rect)
print(f"Rectangle perimeter: {rect.compute_perimeter()}")

# v-4)
tri = Triangle()
print(tri)
print(f"Triangle perimeter: {tri.compute_perimeter()}")
```

Key Concepts Demonstrated:

1. **Inheritance:** Creating class hierarchies (Shape → Polygon → Rectangle/Triangle)
2. **Method Overriding:** Subclasses override parent methods (`__str__`/`toString()`)
3. **Polymorphism:** Same method call produces different results based on object type
4. **Super Keyword:** Calling parent class methods/constructors
5. **Protected Access:** Allowing subclasses to access parent fields
6. **Special Methods:** `__str__` in Python, `toString()` in Java

End of Question 4