SWEN20003
Object Oriented Software Development
Workshop 5

Eleanor McMurtry

Semester 2, 2020

Workshop

Questions

1. Download the chess example from Canvas, and run it in IntelliJ. Make sure you understand how it works. Implement the following additional chess pieces:

   - `Pawn`, which can move two spaces vertically if it hasn't yet moved, otherwise one space vertically
   - `Bishop`, which can move any number of spaces diagonally

   Solution:

```
public class Pawn extends Piece {
    private boolean hasMoved = false;

    public Pawn(int currentRow, int currentColumn) {
        super(currentRow, currentColumn);
    }

    @Override
    public boolean isValidMove(int toRow, int toColumn) {
        int row = super.getCurrentRow();
        int col = super.getCurrentColumn();

        if (row == toRow) {
            if (!hasMoved && Math.abs(col - toColumn) == 2) {
                hasMoved = true;
                return true;
            } else if (hasMoved && Math.abs(col - toColumn) == 1) {
                return true;
            } else {
                return false;
            }
        } else {
            return false;
        }
    }
}

public class Bishop extends Piece {
    public Bishop(int currentRow, int currentColumn) {
        super(currentRow, currentColumn);
    }

    @Override
    public boolean isValidMove(int toRow, int toColumn) {
        int row = super.getCurrentRow();
```

```
        int col = super.getCurrentColumn();

        // Must move the same amount horizontally and vertically
        return Math.abs(toRow - row) == Math.abs(toColumn - col);
    }
}
```

2. Consider the below class.

```
public class Shape {
    public final double x;
    public final double y;

    public Shape(double x, double y) {
        this.x = x;
        this.y = y;
    }

    public double getArea() { return 0; }
    public double getPerimeter() { return 0; }
    public String toString() { return "Plain Shape"; }
}
```

(a) Create the following subclasses:

- `Rectangle`, which has a **width** and **height**
- `Circle`, which has a **radius**

(b) Override the `getArea()` and `getPerimeter()` methods in your subclasses.

(c) Write appropriate `toString()` methods in your subclasses using the overridden methods.

Solution:

```
public class Rectangle extends Shape {
    public final double width;
    public final double height;

    public Rectangle(double x, double y, double width, double height) {
        super(x, y);
        this.width = width;
        this.height = height;
    }

    @Override
    public double getArea() {
        return width * height;
    }

    @Override
    public double getPerimeter() {
        return 2 * (width + height);
    }

    @Override
    public String toString() {
        return String.format("Rectangle at (%f, %f): area = %f, perimeter = %f",
            x, y, getArea(), getPerimeter());
    }
}
```

```
public class Circle extends Shape {
    public final double radius;

    public Rectangle(double x, double y, double radius) {
```

```
        super(x, y);
        this.radius = radius;
    }

    @Override
    public double getArea() {
        return Math.PI * radius * radius;
    }

    @Override
    public double getPerimeter() {
        return 2 * Math.PI * radius;
    }

    @Override
    public String toString() {
        return String.format("Circle at (%f, %f): area = %f, perimeter = %f",
            x, y, getArea(), getPerimeter());
    }
}
```

(d) Make the `getArea()` and `getPerimeter()` methods abstract (in the `Shape` class).

Solution:

```
public abstract class Shape {
    public final double x;
    public final double y;

    public Shape(double x, double y) {
        this.x = x;
        this.y = y;
    }

    public abstract double getArea();
    public abstract double getPerimeter();
    public String toString() { return "Plain Shape"; }
}
```

(e) Write a `main` class. You should create a `Shape` array with 10 shapes (randomly chosen between `Circle` and `Rectangle`), where their parameters are chosen randomly between 0 and 5. Loop over this array and print the result of calling `toString()` on them.

Solution:

```
public class Program {
    private static Shape randomShape() {
        double x = Math.random() * 5;
        double y = Math.random() * 5;
        if (Math.random() < 0.5) {
            double width = Math.random() * 5;
            double height = Math.random() * 5;
            return new Rectangle(x, y, width, height);
        } else {
            double radius = Math.random() * 5;
            return new Circle(x, y, radius);
        }
    }

    public static void main(String[] args) {
        Shape[] shapes = new Shape[10];
        for (int i = 0; i < shapes.length; ++i) {
            shapes[i] = randomShape();
        }
        for (Shape shape : shapes) {
```

```
                System.out.println(shape);
            }
        }
    }
```

3. Consider the below classes.

```
public class HttpRequest {
    private static final short PORT = 80;
    public final String address;
    public final String file;
    public final String method;

    public HttpRequest(String address, String file, String method) {
        this.address = address;
        this.file = file;
        this.method = method;
    }

    public short getPort() {
        return PORT;
    }

    public String getAddress() {
        return address;
    }

    public String getFullRequest() {
        return String.format("%s %s HTTP/1.1\r\n\r\n", method, file);
    }
}
public class FtpRequest {
    private static final short PORT = 21;
    public final String address;
    public final String file;

    public FtpRequest(String address, String file) {
        this.address = address;
        this.file = file;
    }

    public short getPort() {
        return PORT;
    }

    public String getAddress() {
        return address;
    }

    public String getFullRequest() {
        return String.format("RETR %s\x15\x12", file);
    }
}
```

(a) Create a `Request` base class with appropriate methods, including at least one abstract method. Change the above classes to inherit from `Request`.

Solution:

```
public abstract class Request {
    public final String address;
    public final String file;
```

```
    public Request(String address, String file) {
        this.address = address;
        this.file = file;
    }

    public abstract short getPort();

    public String getAddress() {
        return address;
    }

    public abstract String getFullRequest();
}
public class HttpRequest extends Request {
    private static final short PORT = 80;
    public final String method;

    public HttpRequest(String address, String file, String method) {
        super(address, file);
        this.method = method;
    }

    @Override
    public short getPort() {
        return PORT;
    }

    @Override
    public String getFullRequest() {
        return String.format("%s %s HTTP/1.1\r\n\r\n", method, file);
    }
}
public class FtpRequest {
    private static final short PORT = 21;

    public FtpRequest(String address, String file) {
        super(address, file);
    }

    @Override
    public short getPort() {
        return PORT;
    }

    @Override
    public String getFullRequest() {
        return String.format("RETR %s\x15\x12", file);
    }
}
```

(b) Write a `main` method to test your code. It should accept **command-line arguments** (via the `args` parameter of `main`). Your program should accept each of the following formats:

- `http google.com GET /`, in which case it should create an instance of `HttpRequest` with address `google.com`, method `GET`, and file `/`
- `ftp eleanorm.info swen20003.txt`, in which case it should create an instance of `FtpRequest` with address `eleanorm.info` and file `swen20003.txt`

It should create an appropriate instance and upcast it to `Request`, then print the result of calling `getFullRequest()`.

Solution:

```
public class Program {
```

```
    private static Request getRequest(String[] args) {
        if (args.length == 0) {
            System.out.println("Must pass at least one argument.");
            System.exit(-1);
        } else if (args[0].equalsIgnoreCase("http")) {
            if (args.length < 4) {
                System.out.println("Must pass 4 arguments for HTTP.");
                System.exit(-1);
            } else {
                return new HttpRequest(args[1], args[3], args[2]);
            }
        } else if (args[0].equalsIgnoreCase("ftp")) {
            if (args.length < 3) {
                System.out.println("Must pass 4 arguments for FTP.");
                System.exit(-1);
            } else {
                return new FtpRequest(args[1], args[2]);
            }
        }

        return null;
    }

    public static void main(String[] args) {
        Request req = getRequest(args);
        System.out.println(req.getFullRequest());
    }
}
```

4. We will now use inheritance with data structures.

   (a) Define a `Node` class, containing a `public final` integer (its **value**—) and two `Node`s (the **next** and **previous** nodes). Add a getter for each node. (The node may be `null`, meaning there is no next/previous node.)

   Solution:

```
public class Node {
    public final int value;
    private Node next;
    private Node prev;

    public Node(int value) {
        this.value = value;
    }

    public Node getNext() {
        return next;
    }

    public Node getPrev() {
        return prev;
    }
}
```

   (b) Define a `LinkedList` class that contains a node (the **root**). It should have the following methods:

   - `Node root()`: return the root node
   - `int length()`: return the length (i.e. the number of nodes in the list)[1]
   - `void append(int value)`: create a new node with the appropriate value, and add it to the end of the list
   - `void insert(int index, int value)`: create a new node with the appropriate value, and insert it at the index (starting from 0)

[1] Bonus challenge: make this $O(1)$.

- `void remove(int n)`: remove the nth node (starting from 0)

Solution:

```
public class LinkedList {
    private Node root;
    private int length;

    public Node root() {
        return root;
    }

    public int length() {
        return length;
    }

    public void append(int value) {
        // Create the root if necessary
        if (root == null) {
            ++length;
            root = new Node(value);
        } else {
            ++length;
            Node current = root;
            while (current.getNext() != null) {
                current = current.getNext();
            }
            Node next = new Node(value);
            current.setNext(next);
            next.setPrev(current);
        }
    }

    public void insert(int index, int value) {
        // Special-case insertion at root
        if (index == 0) {
            ++length;
            Node node = new Node(value);
            if (root != null) {
                root.setPrev(node);
                node.setNext(root);
            }
            root = node;
        } else {
            int currentIndex = 0;
            Node current = root;
            Node prev = current;
            while (current != null && currentIndex < index) {
                prev = current;
                current = current.getNext();
                ++currentIndex;
            }
            // Check that we reached the target index
            if (currentIndex == index) {
                ++length;
                Node node = new Node(value);
                prev.setNext(node);
                if (current != null) {
                    current.setPrev(node);
                }

                node.setPrev(prev);
```

```
                node.setNext(current);
            }
        }
    }

    public void remove(int index) {
        // Special-case deletion at root
        if (index == 0) {
            if (root != null) {
                root = root.getNext();
                if (root != null) {
                    root.setPrev(null);
                }
                --length;
            }
        } else {
            int currentIndex = 0;
            Node current = root;
            Node prev = current;
            while (current != null && currentIndex < index) {
                prev = current;
                current = current.getNext();
                ++currentIndex;
            }
            // Check that we reached the target index
            if (currentIndex == index) {
                --length;
                if (current != null) {
                    Node next = current.getNext();
                    prev.setNext(next);
                    if (next != null) {
                        next.setPrev(prev);
                    }
                } else {
                    prev.setNext(null);
                }
            }
        }
    }

    public String toString() {
        String result = "[ ";
        Node current = root;
        while (current != null) {
            result += current.value + " ";
            current = current.getNext();
        }
        return result + "]";
    }
}
```

(c) Define a `Stack` class inheriting from `LinkedList`. Add the following methods:

- `void push(int value)`: add a node with the appropriate value at the end of the list
- `Integer pop()`: remove the last node and return its value (or `null` if there is no such node)

Solution:

```
public class Stack extends LinkedList {
    public void push(int value) {
        super.insert(0, value);
    }
```

```
    public Integer pop() {
        if (super.length() > 0) {
            int value = super.root().value;
            super.remove(0);
            return value;
        } else {
            return null;
        }
    }
}
```

(d) Define a `Queue` class inheriting from `LinkedList`. Add the following methods:

- `void enqueue(int value)`: add a node with the appropriate value to the end of the list
- `Integer take()`: remove the first node and return its value (or `null` if there is no such node)

Solution:

```
public class Queue extends LinkedList {
    public void enqueue(int value) {
        super.insert(super.length(), value);
    }

    public Integer take() {
        if (super.length() > 0) {
            int value = super.root().value;
            super.remove(0);
            return value;
        } else {
            return null;
        }
    }
}
```

5. We are ambitious Java enthusiasts and are already ready to begin creating our own small 'graphics' library. Our first task is to **design** a system to render simple shapes onto the screen. For now, we are concerned about two types of shapes in particular: **square**s and **triangle**s. A shape has a specific area associated with it, and it can also be rendered to the screen. **You are not required to implement any rendering logic.** For simplicity, you are to print a description of the shape to be rendered to the console instead of rendering anything to the screen.

   A shape also has a **colour** associated with it. We will be using the the RGB colour system which specifies a colour through three values: *red*, *green*, *blue*. The red, green, and blue values of a colour must be within the range of 0-255 (inclusive) at all times. If a colour is not specified, a shape's default colour is black (red = 0, green = 0, blue = 0). Your `Colour` implementation should be immutable. **The system should be compatible with the following driver class:**

```
public class Driver {

    private static final int MAX_SHAPES = 4;

    public static void main(String[] args) {
        Shape[] shapes = new Shape[MAX_SHAPES];
        // Black rectangle (red=0, green=0, blue=0) with width 20.52px and height 50px
        shapes[0] = new Rectangle(20.52, 50);
        // Crimson-coloured triangle (r=220, g=20, b=60) with base 392.2px and height 0.01px
        shapes[1] = new Triangle(392.2, 0.01, new Colour(220, 20, 60));
        // White (r=255, g=255, b=255) rectangle with width 50px and height 50.3px
        shapes[2] = new Rectangle(50, 50.3, Colour.WHITE);
        // Black triangle (red=0, green=0, blue=0) with base 10px and height 20.12px
        shapes[3] = new Triangle(10, 20.12);

        for (Shape shape : shapes) {
            shape.render();
        }
```

```
        double average = 0;
        for (int i = 0; i < MAX_SHAPES; i++) {
            average = (average * i + shapes[i].getArea()) / (i + 1);
        }
        System.out.format("Average area of rendered shapes: %.2f\n", average);
    }
}
```

Example output:

```
Drawing a Rectangle with colour:(0, 0, 0) and area:1026.00px2
Drawing a Triangle with colour:(220, 20, 60) and area:1.96px2
Drawing a Rectangle with colour:(255, 255, 255) and area:2515.00px2
Drawing a Triangle with colour:(0, 0, 0) and area:100.60px2
Average area of rendered shapes: 910.89px2
```

Solution:

```
public class Colour {
    public static final Colour BLACK = new Colour(0, 0, 0);
    public static final Colour WHITE = new Colour(255, 255, 255);
    public final int r;
    public final int g;
    public final int b;

    public Colour(int r, int g, int b) {
        this.r = r;
        this.g = g;
        this.b = b;
    }

    @Override
    public String toString() {
        return String.format("(%d, %d, %d)", r, g, b);
    }
}

public abstract class Shape {
    public final Colour colour;
    public final String name;

    public Shape(String name) {
        this.name = name;
        this.colour = Colour.BLACK;
    }

    public Shape(String name, Colour colour) {
        this.name = name;
        this.colour = colour;
    }

    public abstract double getArea();

    public void render() {
        System.out.format("Drawing a %s with colour:%s and area: %.2fpx2\n",
            name, colour, getArea());
    }
}

public class Rectangle extends Shape {
```

```
    public final double width;
    public final double height;

    public Rectangle(double width, double height) {
        super("Rectangle");
        this.width = width;
        this.height = height;
    }

    public Rectangle(double width, double height, Colour colour) {
        super("Rectangle", colour);
        this.width = width;
        this.height = height;
    }

    @Override
    double getArea() {
        return width * height;
    }
}

public class Triangle extends Shape {
    public final double base;
    public final double height;

    public Triangle(double base, double height) {
        super("Triangle");
        this.base = base;
        this.height = height;
    }

    public Triangle(double base, double height, Colour colour) {
        super("Triangle", colour);
        this.base = base;
        this.height = height;
    }

    @Override
    double getArea() {
        return 1 / 2.0 * base * height;
    }
}
```