Assignment One

Documentation

Odinakachukwu Nzekwe

Athabasca University

**Computer Science 308:**

Java for Programmers

COMP 308

**# Problem\_1 Documentation**

## Overview

The `problem\_1` package provides a `Circle` class for managing circles in a two-dimensional coordinate system. It includes functionalities for calculating the circumference and area of the circle, setting the radius with a maximum limit, printing circle attributes, checking if a point is inside the circle, and moving the origin.

## Class: Circle

### Description

Represents a circle with properties such as coordinates of the center (`x` and `y`), radius, and methods to perform operations on circles.

### Attributes

- `x`: The x-coordinate of the center of the circle.

- `y`: The y-coordinate of the center of the circle.

- `radius`: The radius of the circle.

- `MAX\_RADIUS`: A constant representing the maximum allowed radius for the circle.

### Constructors

1. `Circle()`: Initializes a circle with default values.

2. `Circle(double x, double y, double radius)`: Initializes a circle with specified coordinates and radius.

### Methods

1. `double circumference()`: Calculates and returns the circumference of the circle.

2. `double area()`: Calculates and returns the area of the circle.

3. `void setRadius(double r)`: Sets the radius of the circle with a maximum limit check.

4. `void printAttributes()`: Prints the coordinates, radius, circumference, and area of the circle.

5. `boolean isInside(double xPoint, double yPoint)`: Checks if a given point is inside the circle.

6. `void move(int xMove, int yMove)`: Moves the origin by the specified amount.

### Usage

1. Create instances of `Circle` using default constructor or specifying coordinates and radius.

2. Use various methods to perform operations such as calculating circumference, checking if a point is inside the circle, etc.

3. Test the functionalities using the `main` method provided in the class.

## Note

- This package offers a convenient way to work with circles in a 2D space, providing essential operations and checks related to circles.

- Users can adjust the maximum allowed radius by modifying the `MAX\_RADIUS` constant as needed.

# **Problem\_2 Documentation**

## Overview

The `problem\_2` package provides classes for managing mailing addresses and generating shipping labels. It includes three classes: `FullName`, `MailingAddress`, and `ShippingLabel`. These classes enable the creation of structured mailing addresses and printing of shipping labels.

## Class: FullName

### Description

Represents a full name with title, first name, middle name, and last name.

### Attributes

- `title`: The title of the person's name (e.g., Mr., Ms.).

- `firstName`: The first name of the person.

- `middleName`: The middle name of the person.

- `lastName`: The last name of the person.

### Constructors

1. `FullName(String title, String firstName, String middleName, String lastName)`: Initializes a FullName object with the provided values.

### Methods

- `toString()`: Overrides the `toString()` method to return a nicely formatted full name.

## Class: MailingAddress

### Description

Represents a mailing address with recipient's full name, street address, city, province, and postal code.

### Attributes

- `fullName`: A FullName object representing the recipient's full name.

- `streetAddress`: The street address of the recipient.

- `city`: The city of the recipient.

- `province`: The province of the recipient.

- `postalCode`: The postal code of the recipient.

### Constructors

1. `MailingAddress(FullName fullName, String streetAddress, String city, String province, String postalCode)`: Initializes a MailingAddress object with the provided values.

### Methods

- `toString()`: Overrides the `toString()` method to return a nicely formatted mailing address.

## Class: ShippingLabel

### Description

Represents a shipping label with ship-from and ship-to addresses.

### Attributes

- `shipFrom`: A MailingAddress object representing the sender's address.

- `shipTo`: A MailingAddress object representing the recipient's address.

### Constructors

1. `ShippingLabel(MailingAddress shipFrom, MailingAddress shipTo)`: Initializes a ShippingLabel object with ship-from and ship-to addresses.

### Methods

- `printLabel()`: Prints the shipping label to the console, including ship-from and ship-to addresses.

## Class: Main

### Description

Demonstrates the usage of the package by creating instances of `FullName`, `MailingAddress`, and `ShippingLabel` classes and printing a shipping label.

### Usage

1. Create instances of `FullName` for sender and receiver.

2. Create instances of `MailingAddress` for sender and receiver.

3. Create a `ShippingLabel` object with ship-from and ship-to addresses.

4. Call the `printLabel()` method to print the shipping label.

## Note

- This package provides a convenient way to manage mailing addresses and generate shipping labels for various applications.

# **Cycle Documentation Problem\_3**

## Overview

This document provides documentation for the `Cycle` hierarchy, which includes the `Cycle` class representing a generic cycle and its subclasses: `Unicycle`, `Bicycle`, and `Tricycle`. Additionally, it covers the `Main` class, which demonstrates polymorphism with cycles.

## Class: Cycle

### Description

Represents a generic cycle with a method to return the number of wheels and another method to print a message indicating riding a cycle with the number of wheels.

### Methods

1. `int wheels()`: Returns the number of wheels for this cycle. Returns 0 for the default implementation.

2. `void ride()`: Prints a message indicating riding a cycle with the number of wheels.

## Class: Unicycle

### Description

Represents a unicycle, extending `Cycle`.

### Methods

1. `int wheels()`: Overrides the `wheels()` method in `Cycle` to return 1, indicating a single wheel.

## Class: Bicycle

### Description

Represents a bicycle, extending `Cycle`.

### Methods

1. `int wheels()`: Overrides the `wheels()` method in `Cycle` to return 2, indicating two wheels.

## Class: Tricycle

### Description

Represents a tricycle, extending `Cycle`.

### Methods

1. `int wheels()`: Overrides the `wheels()` method in `Cycle` to return 3, indicating three wheels.

## Class: Main

### Description

Demonstrates polymorphism with cycles by creating instances of `Unicycle`, `Bicycle`, and `Tricycle`, and storing them in variables of type `Cycle`. Calls the `ride()` method for each instance to observe the polymorphic behavior.

### Usage

1. Create instances of `Unicycle`, `Bicycle`, and `Tricycle`.

2. Store them in variables of type `Cycle`.

3. Call the `ride()` method for each variable to observe polymorphic behavior.

## Note

- This hierarchy demonstrates inheritance and polymorphism in Java, where subclasses override methods of the superclass to provide specific behavior.

* The `Main` class illustrates how polymorphism allows for the flexibility of treating objects of different subclasses uniformly.

**# Problem\_4 Package Documentation**

## Overview

The `problem\_4` package provides classes and interfaces for representing musical instruments and performing operations on them. It includes the `Playable` interface and several classes representing different types of musical instruments.

## Interface: Playable

### Description

Defines a contract for objects that can be played.

### Methods

- `void play()`: A method signature for playing an instrument.

## Class: Instrument

### Description

Represents a generic musical instrument that can be played.

### Implements

- `Playable`: Implements the `play()` method.

### Methods

- `void adjust()`: Method to adjust the instrument.

## Class: Wind

### Description

Represents a wind instrument, extending the `Instrument` class.

### Implements

- `Playable`: Implements the `play()` method.

### Methods

- `void clearSpitValve()`: Method to clear the spit valve of the wind instrument.

## Class: Percussion

### Description

Represents a percussion instrument, extending the `Instrument` class.

### Implements

- `Playable`: Implements the `play()` method.

### Methods

- `void adjust()`: Method to adjust the percussion instrument.

## Class: Stringed

### Description

Represents a stringed instrument, extending the `Instrument` class.

### Implements

- `Playable`: Implements the `play()` method.

### Methods

- `void tune()`: Method to tune the stringed instrument.

## Class: Brass

### Description

Represents a brass instrument, extending the `Wind` class.

### Methods

- `void adjust()`: Method to adjust the brass instrument.

## Class: Woodwind

### Description

Represents a woodwind instrument, extending the `Wind` class.

### Methods

- `void adjust()`: Method to adjust the woodwind instrument.

## Class: Music5

### Description

Contains static methods to interact with musical instruments.

### Methods

- `static void tune(Playable playable)`: Method to play a `Playable` object.

- `static void tuneAll(Playable[] e)`: Method to play an array of `Playable` objects.

### Main Method

- Instantiates various musical instruments and plays them using the `tuneAll` method.

## Note

- This package provides a flexible framework for managing and playing different types of musical instruments.

- The use of interfaces and inheritance allows for polymorphic behavior, enabling uniform treatment of different instrument types.