## Exercise: Book and Author Classes Again ‐ An Array of Objects as an Instance Variable

In the earlier exercise, a book is written by one and only one author. In reality, a book can be written by one or more author. Modify the Book class to support one or more authors by changing the instance variable authors to an Author array. Reuse the Author class written earlier.

Notes:

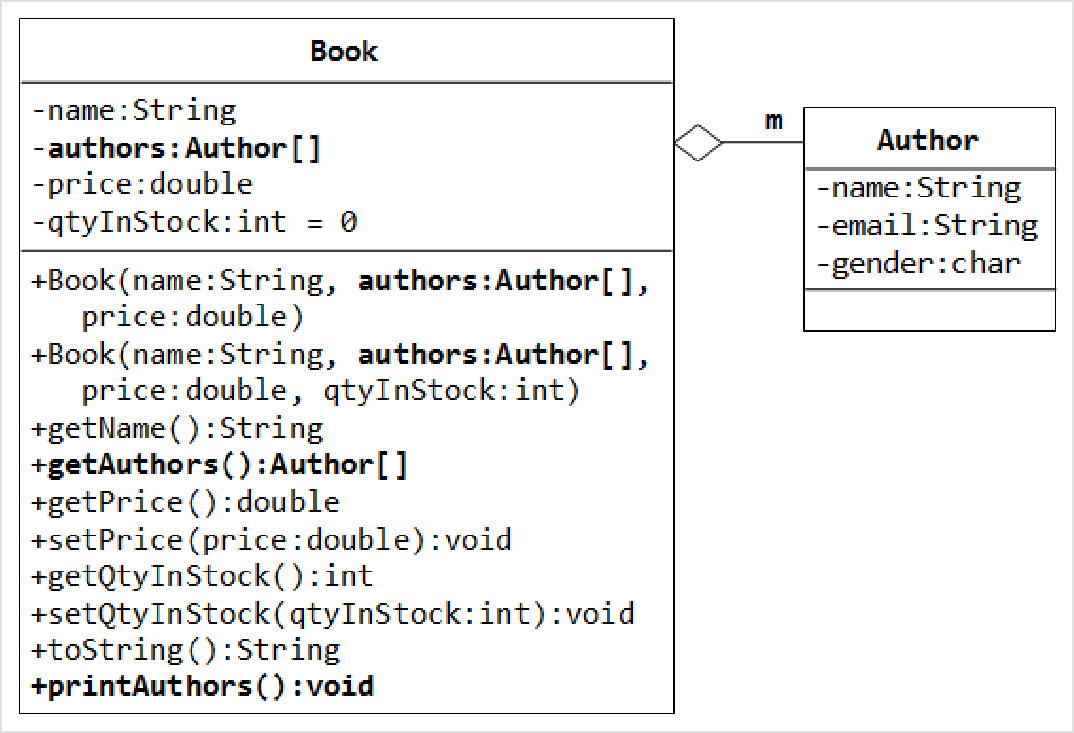
The constructors take an array of Author ﴾i.e., Author[]﴿, instead of an Author instance.

The toString() method shall return "book‐name by *n* authors", where *n* is the number of authors. A new method printAuthors() to print the names of all the authors.

You are required to:

* + 1. Write the code for the Book class. You shall re‐use the Author class written earlier.
    2. Write a test program ﴾called TestBook﴿ to test the Book class.

Hints:



// Declare and allocate an array of Authors Author[] authors = new Author[2];

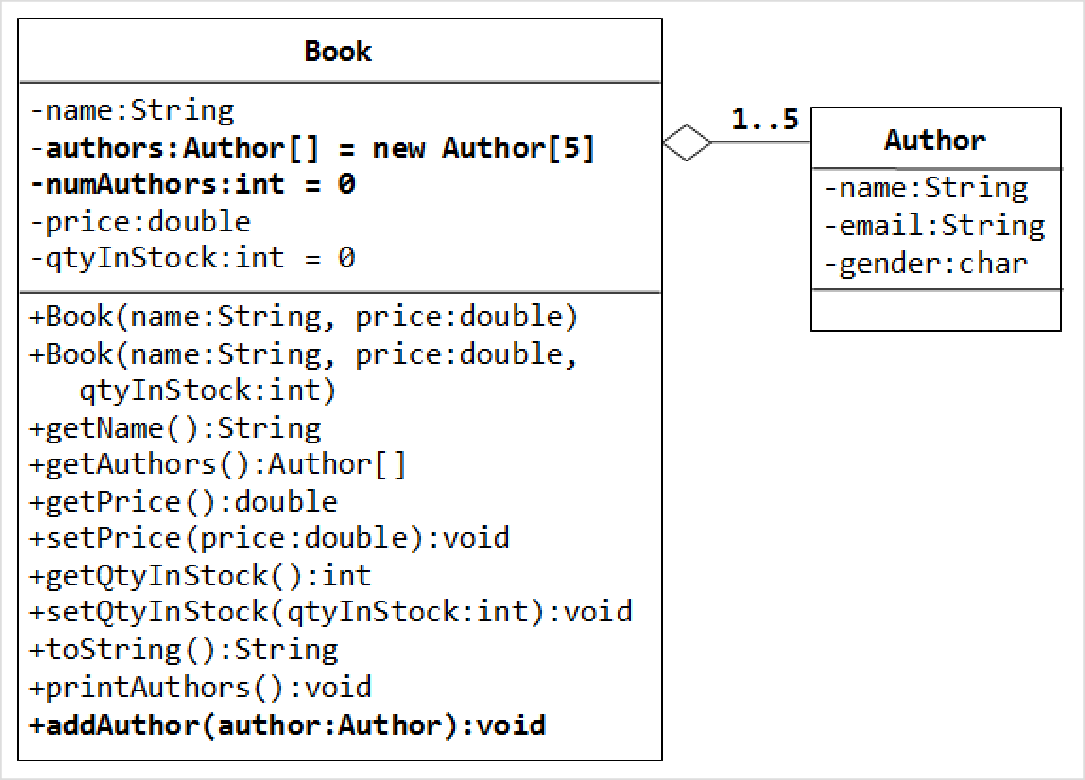
authors[0] = new Author("Tan Ah Teck", "[AhTeck@somewhere.com](mailto:AhTeck@somewhere.com)", 'm'); authors[1] = new Author("Paul Tan", "[Paul@nowhere.com](mailto:Paul@nowhere.com)", 'm');

// Declare and allocate a Book instance

Book javaDummy = new Book("Java for Dummy", authors, 19.99, 99); System.out.println(javaDummy); // toString() System.out.print("The authors are: ");

javaDummy.printAuthors();

## Exercise: Book and Author Classes Once More ‐ A Fixed‐length Array of Objects as an Instance Variable



* In the above exercise, the number of authors cannot be changed once a Book Instance is constructed. Suppose that we wish to allow the user to add more authors ﴾which is really unusual but presented here for academic purpose﴿. We shall remove the authors from the constructors, and add a new method called addAuthor() to add the given Author instance to this Book.
* We also need to pre‐allocate an Author array, with a fixed length ﴾says 5 ‐ a book is written by 1 to 5 authors﴿, and use another instance variable numAuthors ﴾int﴿ to keep track of the actual number of authors.

You are required to:

* + 1. Modify your Book class to support this new requirement.

Hints:

public class Book {

// private instance variable

private Author[] authors = new Author[5]; // declare and allocate the array

// BUT not the element's instance

private int numAuthors = 0;

......

......

public void addAuthor(Author author) { authors[numAuthors] = author;

++numAuthors;

}

}

// Test program

Book javaDummy = new Book("Java for Dummy", 19.99, 99); System.out.println(javaDummy); // toString() System.out.print("The authors are: "); javaDummy.printAuthors();

javaDummy.addAuthor(new Author("Tan Ah Teck", "[AhTeck@somewhere.com](mailto:AhTeck@somewhere.com)", 'm')); javaDummy.addAuthor(new Author("Paul Tan", "[Paul@nowhere.com](mailto:Paul@nowhere.com)", 'm')); System.out.println(javaDummy); // toString()

System.out.print("The authors are: "); javaDummy.printAuthors();

* + 1. Try writing a method called removeAuthorByName(authorName), that remove the author from this Book

instance if authorName is present. The method shall return true if it succeeds.

boolean removeAuthorByName(String authorName)

Advanced Note: Instead of using a fixed‐length array in this case, it is better to be a dynamically allocated array ﴾e.g., ArrayList﴿, which does not have a fixed length.

## Exercise: Bouncing Balls ‐ Ball and Container Classes

A class called Ball is designed as shown in the class diagram.

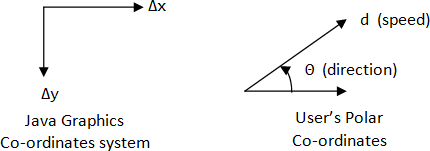
The Ball class contains the following private instance variables:

x, y and radius, which represent the ball's center (x, y) co‐ordinates and the radius, respectively.

xDelta ﴾Δx﴿ and yDelta ﴾Δy﴿, which represent the displacement ﴾movement﴿ per step, in the x and y direction respectively.

The Ball class contains the following public methods:

A constructor which accepts x, y, radius, speed, and direction as arguments. For user friendliness, user specifies speed ﴾in pixels per step﴿ and direction ﴾in degrees in the range of (‐180°, 180°]﴿. For the internal

operations, the speed and direction are to be converted to (Δx, Δy) in the internal representation. Note that the y‐axis of the Java graphics coordinate system is inverted, i.e., the origin (0, 0) is located at the top‐ left corner.

Δx = d × cos(θ) Δy = ‐d × sin(θ)

Getter and setter for all the instance variables.

A method move() which move the ball by one step.

x += Δx y += Δy

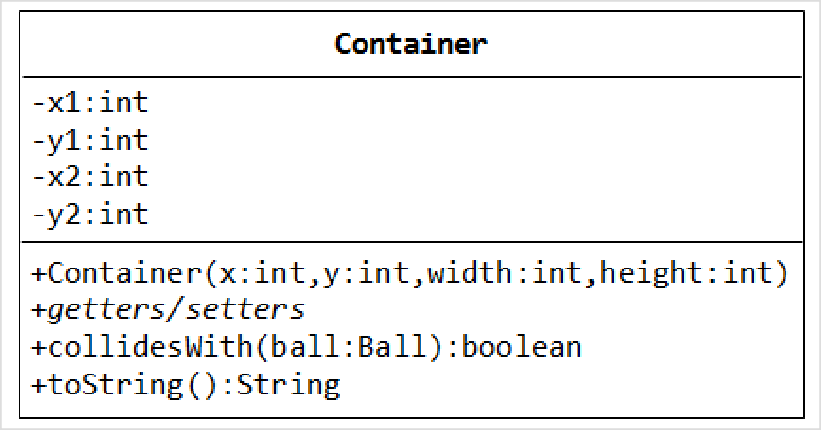
reflectHorizontal() which reflects the ball horizontally ﴾i.e., hitting a vertical wall﴿

Δx = ‐Δx

Δy no changes

reflectVertical() ﴾the ball hits a horizontal wall﴿.

Δx no changes Δy = ‐Δy

toString() which prints the message "Ball at (x, y) of velocity (Δx, Δy)". Write the Ball class. Also write a test program to test all the methods defined in the class.

A class called Container, which represents the enclosing box for the ball, is designed as shown in the class diagram. It contains:

Instance variables (x1, y1) and (x2, y2) which denote the top‐left and bottom‐right corners of the rectangular box.

A constructor which accepts (x, y) of the top‐left corner, width and height as argument, and converts

them into the internal representation ﴾i.e., x2=x1+width‐1﴿. Width and height is used in the argument for safer operation ﴾there is no need to check the validity of x2>x1 etc.﴿.

A toString() method that returns "Container at (x1,y1) to (x2, y2)".

A boolean method called collidesWith(Ball), which check if the given Ball is outside the bounds of the container box. If so, it invokes the Ball's reflectHorizontal() and/or reflectVertical() to change the

movement direction of the ball, and returns true.

public boolean collidesWith(Ball ball) {

if (ball.getX() ‐ ball.getRadius() <= this.x1 || ball.getX() ‐ ball.getRadius() >= this.x2) { ball.reflectHorizontal();

return true;

}

......

}

Use the following statements to test your program:

Ball ball = new Ball(50, 50, 5, 10, 30);

Container box = new Container(0, 0, 100, 100); for (int step = 0; step < 100; ++step) {

ball.move(); box.collidesWith(ball);

System.out.println(ball); // manual check the position of the ball

}

Một lớp được gọi là Container, đại diện cho hộp bao quanh quả bóng, được thiết kế như thể hiện trong sơ đồ lớp. Nó chứa:

Các biến thể hiện (x1, y1) và (x2, y2) biểu thị các góc trên cùng bên trái và dưới cùng bên phải của hộp hình chữ nhật.

Hàm tạo chấp nhận (x, y) của góc trên cùng bên trái, chiều rộng và chiều cao làm đối số và chuyển đổi

chúng vào biểu diễn bên trong ﴾tức là, x2=x1+width‐1﴿. Chiều rộng và chiều cao được sử dụng trong đối số để vận hành an toàn hơn ﴾không cần kiểm tra tính hợp lệ của x2>x1, v.v.﴿.

Phương thức toString() trả về "Vùng chứa tại (x1,y1) thành (x2, y2)".

Một phương thức boolean được gọi là va chạmWith(Ball), kiểm tra xem Quả bóng đã cho có nằm ngoài giới hạn của hộp chứa hay không. Nếu vậy, nó sẽ gọi phương thức phản xạHorizontal() và/hoặc phản xạVertical() của Quả bóng để thay đổi

hướng chuyển động của quả bóng, và trả về true.