Homwork1- Particle Simulation

Student name: Yixin Chen (Yit Xiaang Ztang)
Internet ID: chen9176

Part A: Gravity + Bounce

During this part, I should first define the attributes and methods of particles. As to its attributes, the initial position, radius, and initial velocity should be considered. Thus, I will define these attributes in the class of particle and set them as private. Besides, in order to make things easier, I would put gravity inside of the particle class.
Here we show the code for this:

```
public class Particle{
    private double positionX;
    private double positionY;
    private double radius;
    private double[] velocityVec = new double[2];
    private double gravity;
    public Particle(double xPos, double yPos, double r){
        gravity = -9.8; //gravity acceleration g = 9.8 m/s^2
        if(yPos <= r){
            System.out.println("The y position of particle should be greater
than its radius.");
            System.exit(0);// we do not want to put the particle underground.
        }else {
            this.radius = r;
            this.positionX = xPos;
            this.positionY = yPos;

        }
    }
……
}
```

Besides, we want to make setters and getters for these values.

```
public void setRadius(double r){
        if(r >= 0){
        this.radius = r;
        System.out.println("You have successfully set the radius!");
        }
        else {
        System.out.println("The value of Radius cannot be negative!!! Please
type another r value!");
        }
    }
    public void setGravity(double g){
        this.gravity = g;
        System.out.println("You have successfully set the gravity!");
    }
    public void setPositionX(double px){
        this.positionX = px;
        //System.out.println("You have successfully set the x position!");
    }
    public void setPositionY(double py){
        this.positionY = py;
        //System.out.println("You have successfully set the y position!");
    }


    public void setPosition(double px, double py){
        this.positionX = px;
        this.positionY = py;
        System.out.println("You have successfully set the Position!");
    }
    public void setVelocity(double vx, double vy){
        this.velocityVec[0] = vx;
        this.velocityVec[1] = vy;
        System.out.println("You have successfully set the velocity!");
    }

    public double getX(){
        return this.positionX;
    }
    public double getY(){
        return this.positionY;
    }
    public double getRadius(){
        return this.radius;
    }
    public double[] getVelocity(){
        return this.velocityVec;
    }

    public double getGravity(){
        return this.gravity;
    }
```

Then, we should consider the movement of particles in gravity field. For me, under the guidance of the video, I would like to make an update function for updating positions and velocities for the particle in every loop. I also want the particle can bounce against the ground So it should be defined as this:

```
public void update(double dt){
        //velocity = velocity + acceleration * dt
        velocityVec[1] = velocityVec[1] + gravity * dt;
        //position = position + velocity * dt
        positionX = positionX + velocityVec[0] * dt;
        positionY = positionY + velocityVec[1] * dt;

        if(positionY <= radius){
            positionY = radius;
            velocityVec[1] = -1 * velocityVec[1]*0.8;
        }
    }
```

For now, I have successfully defined the particle that the Part A required. For now, I should move into the simulation part- main function.

Define the initial attributes of the particle and make a for loop to generate a series of its movement properties.

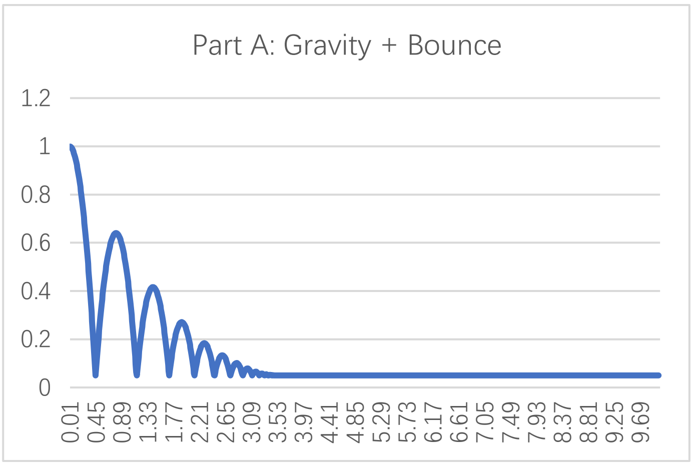
```
public class Simulation {
    public static void main(String[] args) {
        Particle particle = new Particle(0.0, 1.0, 0.05);
        particle.setVelocity(1.0, 0.0);
        System.out.println(particle.getY());

        for(int i = 0; i<1000; i++){
            particle.update(0.01);
            System.out.println(particle.getX() + "," +particle.getY());
        }
    }
}
```

Lets see the outcome of the code and make a plot for this draw in Excel:

You have successfully set the velocity!

```
1.0
0.01,0.99902
0.02,0.9970600000000001
0.03,0.99412
0.04,0.9902
0.05,0.9853
……
9.979999999999832,0.05
9.989999999999831,0.05
9.999999999999831,0.05
```

We can see the outcome is what we really want!



Part B: Obstacles

It should be the same as the particle for defining attributes, setters and getters.

```
public class Obstacle {
    private double obsX;
    private double obsY;
    private double obsRadius;
    private double[] obsPos = new double[2];

    public Obstacle(double x, double y, double r){
        this.obsRadius = r;
        this.obsX = x;
        this.obsY = y;
    }
    public void setRadius(double r){
        if(r >= 0){
        this.obsRadius = r;
        System.out.println("You have successfully set the radius of obstacle!");
        }
        else {
        System.out.println("The value of Radius cannot be negative!!! Please
type another r value!");
        }
    }
    public void setPositionX(double px){
        this.obsX = px;
        System.out.println("You have successfully set the x position of
obstacle!");
    }
    public void setPositionY(double py){
        this.obsY = py;
        System.out.println("You have successfully set the y position of obstacle!");
    }
    public void setPosition(double px, double py){
        this.obsX = px;
        this.obsY = py;
        this.obsPos[0] = this.obsX;
        this.obsPos[1] = this.obsY;
        System.out.println("You have successfully set the Position of obstacle!");
    }
    public double getX(){
        return this.obsX;
    }
    public double getY(){
        return this.obsY;
    }
    public double getRadius(){
        return this.obsRadius;
    }
```

The most interesting and exciting point should be the collision with the particle. For obstacle class, we should first check if they get collided or not and if the particle does hit the obstacle, then we can jump into the collide function public double[] collide(Particle p).

Finally, we should add another item public void handleCollision(Obstacle obstacle)
 for particle to collide with the obstacle.

During the video tutor, we know how the 2D collision function for objects and we combine them together.

For Obstacle class:

```
    public boolean checkCollision(Particle p){
        double distance = Vector2DMath.magnitude(obsX - p.getX(), obsY - p.getY());
        return distance <= obsRadius +p.getRadius();
        }

    public double[] collide(Particle p){
        double[] normal = Vector2DMath.normal(p.getX() - this.obsX, p.getY() - this.obsY);
        double distance = obsRadius + p.getRadius();
        p.setPositionX(obsX + normal[0] * distance);
        p.setPositionY(obsY + normal[1] * distance);
        return normal;
    }
```

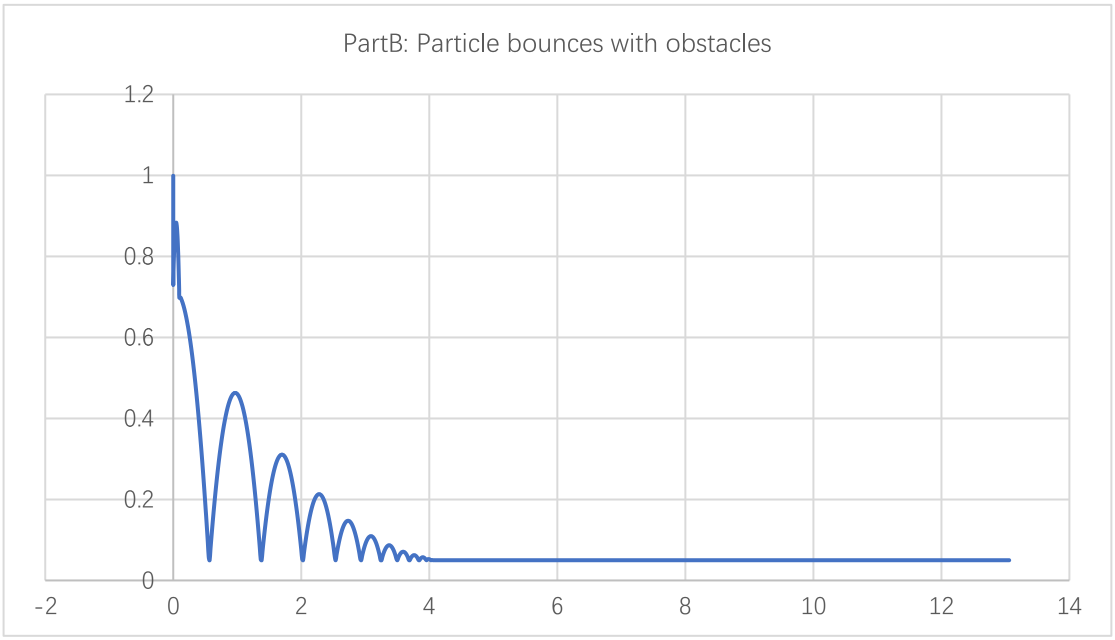
For Particle class:

```
    public void handleCollision(Obstacle obstacle){

        double[] normal = new double[2];
        if(obstacle.checkCollision(this)){
            normal = obstacle.collide(this);

            //bounce (reflect)
            //if the velocity is too small to bounce (|v| < 0.2)
            if(Vector2DMath.magnitude(velocityVec[0], velocityVec[1]) < 0.2){

                velocityVec[0] = normal[0] * 0.5;
                velocityVec[1] = normal[1] * 0.5;
            }else{
            double[]  reflect  =  Vector2DMath.reflect(normal,  velocityVec[0],
velocityVec[1]);
            velocityVec[0] = reflect[0] * 0.75;
            velocityVec[1] = reflect[1] * 0.75;
        }
        }
    }
```

Then, we can generate the collision process by set the particle velocity = 0 and introduces obstacle in Simulation.java.

```
public class Simulation {
    public static void main(String[] args) {
        Particle particle = new Particle(0.0, 1.0, 0.05);
        particle.setVelocity(0.0, 0.0);
        System.out.println(particle.getY());

        Obstacle obstacle = new Obstacle(-0.01, 0.6, 0.1);

        for(int i = 0; i<1000; i++){
            particle.handleCollision(obstacle);
=
            particle.update(0.01);
            System.out.println(particle.getX() + "," +particle.getY());
        }
    }
}
```

We make the whole points together to generate the plot:



It looks nice for its shape and the particle has the collision with the obstacle.