package shape;

public class Circle extends TwoDimensionalShape{

public Circle(Point newCenter, double newRadius) {

center = newCenter;

radius = newRadius;}

@Override

public void move (int addx, int addy, int addz) {

center.x += addx;

center.y += addy;

center.z += addz;

}

@Override public String toString() {

String s = "Shape " + Integer.toString(shapeID) + ": Circle with center ("

+ Double.toString(center.x) + ", " + Double.toString(center.y) +

", " + Double.toString(center.z) + ") and radius " +

Double.toString(radius);

return s;

}

}

package shape;

public class Line extends OneDimensionalShape {

public Line(Point point1, Point point2) {

location1 = point1;

location2 = point2;

}

@Override

public void move (int addx, int addy, int addz) {

location1.x += addx;

location1.y += addy;

location1.z += addz;

location2.x += addx;

location2.y += addy;

location2.z += addz;

}

@Override public String toString() {

String s = "Shape " + Integer.toString(shapeID) + ": Line from "

+ "(" + Double.toString(location1.x) + ", " +

Double.toString(location1.y) + ", " +

Double.toString(location1.z) + ") to (" + Double.toString(location1.x)

+ ", " + Double.toString(location1.y) + ", " +

Double.toString(location1.z) + ")";

return s;

}

}

package shape;

public class OneDimensionalShape extends Shape {

Point location1, location2;

public double getLength() {

double rx = Math.pow((location2.getX() - location1.getX()), 2);

double ry = Math.pow((location2.getY() - location1.getY()), 2);

double rz = Math.pow((location2.getZ() - location1.getZ()), 2);

return Math.sqrt(rx + ry + rz);

}

}

package shape;

public class Point extends ZeroDimensionalShape{

double x ,y, z;

public Point(int newx, int newy, int newz) {

x = newx;

y = newy;

z = newz;

}

public void setCoordinates(int setX, int setY, int setZ) {

x = setX;

y = setY;

z = setZ;

}

public double getX() {

return x;

}

public double getY() {

return y;

}

public double getZ() {

return z;

}

@Override

public void move (int addx, int addy, int addz) {

x += addx;

y += addy;

z += addz;

}

@Override public String toString() {

String s = "Shape " + Integer.toString(shapeID) + ": Point (" + Double.toString(x) +

", " + Double.toString(y) + ", " + Double.toString(z) + ")";

return s;

}

}

package shape;

public class Shape {

int shapeID;

static int count;

public Shape() {

shapeID = count;

count ++;

}

public int getID() {

return shapeID;

}

@Override public String toString() {

String s = Integer.toString(shapeID);

return s;

}

public void move(int x, int y, int z) {

}

}

package shape;

public class Sphere extends ThreeDimensionalShape {

public Sphere(Point newCenter, double newRadius) {

center = newCenter;

radius = newRadius;

}

@Override

public void move (int addx, int addy, int addz) {

center.x += addx;

center.y += addy;

center.z += addz;

}

@Override public String toString() {

String s = "Shape " + Integer.toString(shapeID) + ": Sphere with center ("

+ Double.toString(center.x) + ", " + Double.toString(center.y) +

", " + Double.toString(center.z) + ") and radius " +

Double.toString(radius);

return s;

}

}

package shape;

public class ThreeDimensionalShape extends Shape {

Point center;

double radius;

public double getArea() {

return 4\*Math.PI\*radius\*radius;

}

public double getVolume() {

return ((4\*Math.PI\*radius\*radius\*radius)/3);

}

}

package shape;

public class TwoDimensionalShape extends Shape {

Point center;

double radius;

public double getArea() {

return Math.PI\*radius\*radius;

}

}

package shape;

public class ZeroDimensionalShape extends Shape {

}

package shape;

public class ShapeTester {

public static void main(String[] args) {

Shape[] s = new Shape[4];

s[0] = new Point(1,1,1);

s[1] = new Line(new Point(1,2,3), new Point(3,4,5));

s[2] = new Circle(new Point(1,2,3), 2);

s[3] = new Sphere(new Point(2,2,2), 3);

System.out.println("Shapes:");

for (int i = 0; i < s.length; i++) {

System.out.println(s[i]);

}

System.out.println("\nMove 2,2,2\nShapes:");

for (int i = 0; i<s.length; i++) {

if (s[i] instanceof OneDimensionalShape) {

OneDimensionalShape ods = (OneDimensionalShape)s[i];

System.out.printf("%s length is %f\n", ods, ods.getLength());

}

if (s[i] instanceof TwoDimensionalShape) {

TwoDimensionalShape tds = (TwoDimensionalShape)s[i];

System.out.printf("%s area is %f\n", tds, tds.getArea());

}

if (s[i] instanceof ThreeDimensionalShape) {

ThreeDimensionalShape tds = (ThreeDimensionalShape)s[i];

System.out.printf("%s area is %f\n", tds, tds.getArea());

System.out.printf("%s volume is %f\n", tds, tds.getVolume());

}

}

}

}

run:

Shapes:

Shape 0: Point (1.0, 1.0, 1.0)

Shape 3: Line from (1.0, 2.0, 3.0) to (1.0, 2.0, 3.0)

Shape 5: Circle with center (1.0, 2.0, 3.0) and radius 2.0

Shape 7: Sphere with center (2.0, 2.0, 2.0) and radius 3.0

Move 2,2,2

Shapes:

Shape 3: Line from (1.0, 2.0, 3.0) to (1.0, 2.0, 3.0) length is 3.464102

Shape 5: Circle with center (1.0, 2.0, 3.0) and radius 2.0 area is 12.566371

Shape 7: Sphere with center (2.0, 2.0, 2.0) and radius 3.0 area is 113.097336

Shape 7: Sphere with center (2.0, 2.0, 2.0) and radius 3.0 volume is 113.097336

BUILD SUCCESSFUL (total time: 0 seconds)

Lab3 RANDOM WALKER

package randomwalker;

public class RandomEvent {

int direction;

RandomWalker w;

public RandomEvent(int d, RandomWalker walker) {

direction = d;

w = walker;

}

public RandomWalker getWalker(){

return w;}

public int getDirection(){

return direction;

}

}

package randomwalker;

public class RandomWalkObserver {

int position = 0, numberOfSteps = 0, threshold = 0;

public void moved(RandomEvent e) {

position += e.direction;

numberOfSteps++;

if ((position >= threshold) || (position <= (threshold \* -1))) {

e.getWalker().removeObserver(this);

System.out.println("The walker took " + Integer.toString(numberOfSteps)

+ " steps to get " + Integer.toString(threshold) + " steps away from its starting point");

}

}

}

package randomwalker;

import java.util.ArrayList;

public class RandomWalker {

public ArrayList<RandomWalkObserver> observerList = new ArrayList();

ArrayList<RandomWalkObserver> removeList = new ArrayList();

void addObserver(RandomWalkObserver o) {

observerList.add(o);

}

void removeObserver(RandomWalkObserver o) {

removeList.add(o);

}

int location = 0;

void walk() {

while (observerList.size() > 0) {

RandomEvent event;

double r = Math.random();

if (r < .5) {

event = new RandomEvent(-1, this);

} else {

event = new RandomEvent(1, this);

}

for (RandomWalkObserver o : observerList) {

o.moved(event);

}

for (RandomWalkObserver o : removeList) {

observerList.remove(o);

}

removeList.clear();

//try {

// Thread.sleep(1000);

//} catch (InterruptedException e) {

// System.exit(0);

//}

}

}

}

package randomwalker;

public class WalkTester {

public static void main (String[] args) {

RandomWalker r = new RandomWalker();

RandomWalkObserver a = new RandomWalkObserver();

a.threshold = 5;

r.addObserver(a);

RandomWalkObserver b = new RandomWalkObserver();

b.threshold = 10;

r.addObserver(b);

RandomWalkObserver c = new RandomWalkObserver();

c.threshold = 15;

r.addObserver(c);

RandomWalkObserver d = new RandomWalkObserver();

d.threshold = 20;

r.addObserver(d);

RandomWalkObserver e = new RandomWalkObserver();

e.threshold = 25;

r.addObserver(e);

r.walk();

}

}

Out put

run:

The walker took 11 steps to get 5 steps away from its starting point

The walker took 26 steps to get 10 steps away from its starting point

The walker took 581 steps to get 15 steps away from its starting point

The walker took 604 steps to get 20 steps away from its starting point

The walker took 629 steps to get 25 steps away from its starting point

BUILD SUCCESSFUL (total time: 0 seconds)

Lab 3 thermostat

public class ThermoStat1 {

double minTemp,maxTemp, currentTemp;

public ThermoStat1(int a, int b){

minTemp = a;

maxTemp = b;

}

public void setTemp(double newTemp){

try {

if(newTemp < minTemp){

throw new TempTooLow();

} else if (newTemp > maxTemp){

throw new TempTooHigh();

} else{

currentTemp = newTemp;

System.out.println("The temp is now " + currentTemp);

}

} catch(TempTooLow e){

System.out.println(e);

} catch (TempTooHigh e){

System.out.println(e);

}

}

}

public class TempOutofRange extends Exception {

public TempOutofRange(){

super("Temperature out of range");

}

public TempOutofRange(String s){

super(s);

}

public class TempTooHigh extends TempOutofRange {

public TempTooHigh(){

super("Temp too high");

}

}public class TempTooLow extends TempOutofRange {

TempTooLow(){

super("Temp too low");

}

}

public class ThermostatTester {

public static void main(String[] args) {

ThermoStat1 t = new ThermoStat1(0,100);

t.setTemp(20);

t.setTemp(150);

t.setTemp(-50);

}

}

public class ThermostatTester {

public static void main(String[] args) {

ThermoStat1 t = new ThermoStat1(0,100);

t.setTemp(20);

t.setTemp(150);

t.setTemp(-50);

}

}

Lab 3 enumeration

public interface Enumeration {

//returns true if another element in the collection exists

public boolean hasNext();

//returns next element in the collection as an object

public Object getNext();

}

public class NameCollection {

String[] names;

NameCollection(String[] names)

{

this.names = names;

}

public Enumeration getEnumeration()

{

Enumeration i = new Enumeration()

{

int current = 0;

@Override

public boolean hasNext() {

return (current<names.length);

}

@Override

public String getNext() {

return names[current++];

}

};

return I;}}

public class EnumerationTester {

public static void main (String[] args) {

String[] friends = {"billy", "mark", "Jerry"};

NameCollection t = new NameCollection(friends);

Enumeration i = t.getEnumeration();

for (int j = 0; j < friends.length; j++) {

if (i.hasNext()) {

System.out.println(i.getNext());

}

}

}}

run:

billy

mark

Jerry

BUILD SUCCESSFUL (total time: 0 seconds)

Lab 3 5

public class Item {

String s;

int p;

Item(String string, int priority) {

s = string;

p = priority;

}

@Override public String toString() {

return "String: " + s + " Priority: " + String.valueOf(p);

}

}

import java.util.ArrayList;

public class PriorityQueue {

ArrayList<Item> Q = new ArrayList();

public void add(String s, int p) {

Item a = new Item(s, p);

Q.add(a);

}

public Item remove() {

int topPriority = 0;

int topPriorityIndex = 0;

for (int i = 0; i < Q.size(); i ++) {

if (Q.get(i).p > topPriority) {

topPriority = Q.get(i).p;

topPriorityIndex = i;

}

}

Item result = new Item(Q.get(topPriorityIndex).s, Q.get(topPriorityIndex).p);

Q.remove(topPriorityIndex);

return result;

}

}

run:

billy

mark

Jerry

BUILD SUCCESSFUL (total time: 0 seconds)

**Fin**