PJ02 – Handout

- Description
- Instructions
- Testing
- Submit

Description

For this project, you will be writing a group of classes that can be used to build 3 dimensional geometries. You will create points, build vectors, find surface normals, make faces from triangles, and finally construct a mesh representation of a cube. Some of the principles in this project should be familiar to you, like Cartesian coordinates. Others may be less familiar, such as the cross product of two vectors, in these situations the equations are provided for you. You will learn more about vector cross products in multivariate calculus and linear algebra. For this project an in-depth understanding is not required. If you would like a refresher on Cartesian coordinates, mathinsight.org has an interactive article covering the topic.

This project is worth 5% of your final grade. We recommend that you take it, along with the other projects in the class, very seriously.

You will be implementing five classes: `Point`, `UnitVector`, `Triangle`, `Face`, and `Cube`.

A sixth class `GeoFactory` has been provided for testing.

Note: 5 points of your grade is based on Coding Style. You will need to update the Starter Code to follow the standards described on Brightspace. Use the "Run" button to check your Coding Style without using a submission.

Instructions

Follow the instructions below for each class.

Point.java

This class represents a point in 3 dimensional Cartesian space.

Fields

Field Name	Type	Access Modifier	Description
x	double	private	The x coordinate in Cartesian space
y	double	private	The y coordinate in Cartesian space
z	double	private	The z coordinate in Cartesian space

Constructor

Access Modifier	Constructor Name	Input Parameters	Description
public	Point	double x, double y, double z	Construct a newly allocated **Point** object and instantiate the fields to their respective parameters.

public	Point	None	Construct a newly allocated **Point** object and instantiate all fields set to 0.0.

Methods

Method Name	Return Type	Access Modifier	Input Parameters	Description
getX	double	public	none	Returns the **x** value of this **Point**.
getY	double	public	none	Returns the **y** value of this **Point**.
getZ	double	public	none	Returns the **z** value of this **Point**.
setX	void	public	double x	Sets the **x** value of this **Point**
setY	void	public	double y	Sets the **y** value of this **Point**
setZ	void	public	double z	Sets the **z** value of this **Point**
equals	boolean	public	Point point	Compares this **Point** to point. Return true if **ALL** of the Points values are equal to this Points values. For this assignment double variables are considered equal if they are within +/– 0.0001 precision of each other, see note at the end. Otherwise return false.
toString	String	public	None	Returns the String representation of this **Point**. For Example, given the following fields: • **x** = 1.000 • y = 2.000 • **z =** 0.000 The result of calling **toString()** would be: • (x1.000, y2.000, z0.000) Note the returned String should be formatted to **EXACTLY** 3 decimal places.

UnitVector.java

This class represents a Unit Vector in 3 dimensional Cartesian space. A unit vector is a direction with magnitude 1.

Fields

Field Name	Type	Access Modifier	Description
i	double	private	the **i** component of a vector in 3D space
j	double	private	The **j** component of a vector in 3D space
k	double	private	The **z** component of a vector in 3D space

Constructors:

Access Modifier	Constructor Name	Input Parameters	**Description**
public	UnitVector	double i, double j, double k	Construct a newly allocated **UnitVector** object and instantiate the fields to the specified parameters. Confirm that the magnitude of the UnitVector is equal to 1.000. For this assignment double variables are considered equal if they are within +/– 0.0001 precision of each other, see note at the end. $Magnitude = \sqrt{i^2 + j^2 + k^2}$ If the value is not equal to 1.000 then scale the vector by its magnitude with the following series of equations: $i = \frac{i}{Magnitude}$ $j = \frac{j}{Magnitude}$ $k = \frac{k}{Magnitude}$ Note: in the case where the magnitude is equal to 0 all fields should be initialized to 0.000.

public	UnitVector	Point start, Point end	Construct a newly allocated **UnitVector** object from the two given points using the following equation: i = end.x – start.x j = end.y – start.y k = end.z – start.z Confirm that the magnitude of the UnitVector is equal to 1.000. For this assignment double variables are considered equal if they are within +/– 0.0001 precision of each other, see note at the end. $Magnitude = \sqrt{i^2 + j^2 + k^2}$ If the value is not equal to 1.000 then scale the vector by its magnitude with the following series of equations: $i = \frac{i}{Magnitude}$ $j = \frac{j}{Magnitude}$ $k = \frac{k}{Magnitude}$ Note: in the case where the magnitude is equal to 0 all fields should be initialized to 0.000.
public	UnitVector	none	Construct a newly allocated **UnitVector** object with all fields instantiated to 0.000. (An invalid vector)

Methods:

Method Name	Return Type	Access Modifier	Input Parameters	Description
getI note: 'I' is a capital 'i'	double	public	None	Returns the **i** value of this **UnitVector**

getJ	double	public	None	Returns the **j** value of this **UnitVector**
getK	double	public	None	Returns the **k** value of this **UnitVector**
equals	boolean	public	UnitVector vector	Compares this **UnitVector** to vector. Return true if **ALL** of the Unitvector's values are equal to this UnitVector's values. For this assignment double variables are considered equal if they are within +/− 0.0001 precision of each other, see note at the end. Otherwise return false.
crossProduct	UnitVector	public	UnitVector b	Returns a newly allocated **UnitVector** object with fields set by the following equations: $i = this.j * b.k - this.k * b.j$ $j = this.k * b.i - this.i * b.k$ $k = this.i * b.j - this.j * b.i$ Confirm that the magnitude of the UnitVector is equal to 1.000. For this assignment double variables are considered equal if they are within +/− 0.0001 precision of each other, see note at the end. $Magnitude = \sqrt{i^2 + j^2 + k^2}$ If the value is not equal to 1.000 then scale the vector by its magnitude with the following series of equations: $i = \frac{i}{Magnitude}$ $j = \frac{j}{Magnitude}$ $k = \frac{k}{Magnitude}$ Note: in the case where the magnitude is equal to 0 all fields should be initialized to 0.000.
toString	String	public	None	Returns the String representation of this **UnitVector**. For Example, given the following fields: • **i** = 0.500 • **j** = 0.250 • **k** = 0.250

				The result of calling **toString()** would be: - <0.816i, 0.408j, 0.408k> Note the returned String should be formatted to **EXACTLY** 3 decimal places. If **i**, **j**, and **k**, are equal to 0.000 then the resulting **toString()** should be: - <InvalidUnitVector> For this assignment double variables are considered equal if they are within +/- 0.0001 precision of each other, see note at the end.

Triangle.java

This class represents a Triangle in 3 dimensional Cartesian space. A Triangle is comprised of 3 Vertices **(Points)** and a "Surface Normal" in this case a **UnitVector**. The UnitVector indicates the direction the primary surface of the Triangle is facing.

Fields

Field Name	Type	Access Modifier	Description
vertexA	Point	private	The first vertex of this **Triangle**.
vertexB	Point	private	The second vertex of this **Triangle**.
vertexC	Point	private	The third vertex of this **Triangle**.
surfaceNormal	UnitVector	private	The surface normal of this **Triangle**.

Constructor

Access Modifier	Constructor Name	Input Parameters	Description
public	Triangle	Point vertA, Point vertB, Point vertC	Construct a newly allocated **Triangle** object and instantiate the fields to their respective parameters. Instantiate the unit vector to be the cross product of the vector from A to B (Start at A and go to B) and the vector from A to C (Start at A and go to C). $\vec{AB}.crossProduct(\vec{AC});$ Note: the order matters.
public	Triangle	none	Construct a newly allocated **Triangle** object and instantiate the fields to the following: All vertices should be (x0.000, y0.000, z0.000) The surfaceNormal should be invalid (all fields 0.000)

Methods

Method Name	Return Type	Access Modifier	Input Parameters	Description
getVertexA	Point	public	None	Returns the **Point** representing the Vertex A of this **Triangle**.
getVertexB	Point	public	None	Returns the **Point** representing the Vertex B of this **Triangle**.
getVertexC	Point	public	None	Returns the **Point** representing the Vertex C of this **Triangle**.
getSurfaceNormal	UnitVector	public	None	Returns the **UnitVector** representing the Surface Normal of this **Triangle**.
getVertices	Point[]	public	none	Returns an array of points representing this Triangle. the array index 0 should be vertexA, 1 should be vertexB, and 2 should be vertexC.
equals	boolean	public	Triangle triangle	Compare this Triangle to Triangle triangle. If ALL vertices and surfaceNormal are equal then return true, otherwise return false. For this assignment double variables are considered equal if they are within +/- 0.0001 precision of each other, see note at the end.
toString	String	public	None	Returns the String representation of this **Triangle**. For Example, given the following fields: - **vertexA**= (x0.000, y0.000, z1.000) - **vertexB**= (x1.000, y-1.000, z1.000) - **vertexC**= (x1.000, y0.000, z1.000) - **surfaceNormal** = <0.000i, 0.000j, 1.000k> The result of calling **toString()** would be: - [A(x0.000, y0.000, z1.000); B(x1.000, y-1.000, z1.000); C(x1.000, y0.000, z1.000); N<0.000i, 0.000j, 1.000k>] If the Triangle does not contain 3 unique vertices or the unit vector is invalid the **toString()** should return:

				- [InvalidTriangle]

Face.java

This class represents a single face of a cube in 3 dimensional Cartesian space. (A square) A Square is comprised of 2 Triangles that share 2 Vertices **(Points)** and have identical "Surface Normals" (their **UnitVectors** point the same direction**)**.

Fields

Field Name	Type	Access Modifier	Description
mesh	Triangle[]	private	An array of Triangles that combine to make this **Face** (a bounded square).
surfaceNormal	UnitVector	private	The surface normal of this **Face**.

Constructor

Access Modifier	Constructor Name	Input Parameters	Description
public	Face	Triangle one, Triangle two,	If triangle one and two share two common vertices and their surface normals are equal, construct a newly allocated **Face** object and instantiate the fields to their respective parameters. Triangle one should be index 0 of **mesh**. If the two triangles do not share at least two vertices or the surface normals are not equal, then each triangle should be set to Triangle() and the unit vector set to invalid (all fields 0.000).
public	Face	none	Construct a newly allocated **Face** object and instantiate the fields as follows: each triangle should be set to Triangle() and the unit vector set to invalid (all fields 0.000). (The face is invalid)

Methods

Method Name	Return Type	Access Modifier	Input Parameters	Description
getMesh	Triangle[]	public	None	Returns the **Triangle** array that makes up the **mesh** of this **Face**.
getSurfaceNormal	UnitVector	public	None	Returns the **UnitVector** representing the **surfaceNormal** of this **Face**.

equals	boolean	public	Face face	Compares this **Face** to face. Return true if the Triangles share the same vertices and normal vectors. For this assignment double variables are considered equal if they are within +/– 0.0001 precision of each other, see note at the end. Otherwise return false.
toString	String	public	None	Returns the String representation of this **Triangle**. For Example, given the following fields: **Triangle[0]:** • **vertexA**= (x0.000, y0.000, z1.000) • **vertexB**= (x1.000, y–1.000, z1.000) • **vertexC**= (x1.000, y0.000, z1.000) • **surfaceNormal** = <0.000i, 0.000j, 1.000k> **Triangle[1]:** • **vertexA**= (x0.000, y0.000, z1.000) • **vertexB**= (x0.000, y–1.000, z1.000) • **vertexC**= (x1.000, y–1.000, z1.000) • **surfaceNormal** = <0.000i, 0.000j, 1.000k> **surfaceNormal** = <0.000i, 0.000j, 1.000k> The result of calling **toString()** would be: • {F[A(x0.000, y0.000, z1.000); B(x1.000, y–1.000, z1.000); C(x1.000, y0.000, z1.000)][A(x0.000, y0.000, z1.000); B(x0.000, y–1.000, z1.000); C(x1.000, y–1.000, z1.000)] N<0.000i, 0.000j, 1.000k>} Note: The surface normal for the triangles are not printed, only the one for the face. Note: This is one continuous string with no new line characters. If any Triangle is invalid the result of **toString()** should be: • {InvalidFace}

Cube.java

This class represents a Square in 3 dimensional Cartesian space. A Cube is comprised of 6 **Faces** (squares) where each face shares an edge (has 2 common vertices) with 4 other faces and faces that do not share an edge have opposite surface normal vectors (One face has the inverse **UnitVector** of the other).

Fields

Field Name	Type	Access Modifier	Description
mesh	Face[]	private	An array of **Faces** that make up the **mesh** of the **Cube**.

Constructor

Access Modifier	Constructor Name	Input Parameters	Description
public	Cube	Face one, Face two, Face three, Face four, Face five, Face six	Construct a newly allocated **Cube** object and instantiate the fields to their respective parameters. Confirm that each face shares exactly one edge with each of 4 other faces. Confirm that no face is the same as another face. Confirm that each surfaceNormal of opposed Faces (faces that do not share an edge) is pointing in an opposite direction. If any of the above is false, set each face in mesh to be an invalid face. (all fields 0.000). Note: mesh[0] should be instantiated to Face one and so on to mesh[5] being Face six.
public	Cube	none	Construct a newly allocated **Cube** object and instantiate each **Face** in the **mesh** array to be invalid, all values are 0.000.

Methods

Method Name	Return Type	Access Modifier	Input Parameters	Description
getMesh	Face[]	public	None	Returns the **Face** array representing the entire surface **mesh** of the **Cube**.
toString	String	public	None	Returns the String representation of this **Cube**. The output should be in the format of: "|C" + the toString() of each face. + "|" Example: |C{F[A(x0.000, y0.000, z1.000); B(x1.000, y–1.000, z1.000);

				C(x1.000, y0.000, z1.000)] [A(x0.000, y0.000, z1.000); B(x0.000, y-1.000, z1.000); C(x1.000, y-1.000, z1.000)] N<-0.000i, 0.000j, 1.000k>} {F[A(x0.000, y-1.000, z1.000); B(x0.000, y0.000, z1.000); C(x0.000, y0.000, z0.000)] [A(x0.000, y-1.000, z1.000); B(x0.000, y0.000, z0.000); C(x0.000, y-1.000, z0.000)] N<-1.000i, -0.000j, -0.000k>} {F[A(x1.000, y-1.000, z1.000); B(x0.000, y-1.000, z1.000); C(x0.000, y-1.000, z0.000)] [A(x1.000, y-1.000, z1.000); B(x0.000, y-1.000, z0.000); C(x1.000, y-1.000, z0.000)] N<-0.000i, -1.000j, 0.000k>} {F[A(x1.000, y-1.000, z0.000); B(x0.000, y0.000, z0.000); C(x1.000, y0.000, z0.000)] [A(x1.000, y-1.000, z0.000); B(x0.000, y-1.000, z0.000); C(x0.000, y0.000, z0.000)] N<-0.000i, -0.000j, -1.000k>} {F[A(x1.000, y0.000, z0.000); B(x1.000, y0.000, z1.000); C(x1.000, y-1.000, z1.000)] [A(x1.000, y0.000, z0.000); B(x1.000, y-1.000, z1.000); C(x1.000, y-1.000, z0.000)] N<1.000i, 0.000j, -0.000k>} {F[A(x0.000, y0.000, z0.000); B(x0.000, y0.000, z1.000); C(x1.000, y0.000, z1.000)] [A(x0.000, y0.000, z0.000); B(x1.000, y0.000, z1.000); C(x1.000, y0.000, z0.000)] N<-0.000i, 1.000j, -0.000k>}| Note: This is one continuous string with no new line characters. If any face is invalid, the toString() method should return: - |InvalidCube|

GeoFactory.java

We have provided you with a fully implemented menu to test your classes. You should not modify this program. All of the class you write for this project should function with `GeoFactory.java`. We suggest you read through and familiarize yourself with this program. It handles incorrect inputs on all menus. It does not handle incorrect inputs for doubles when entering data, be careful. Techniques to handle errors such as these will be covered later in this course, but for now only enter correct doubles when prompted.

Additional Notes:

- The only starter code provided is a finished program called GeoFactory.java.
- Order matters, this handout gives specific orders for which variables should be processed and stored.
- There are no added spaces in the Cube.toString() method.
- All double values should be formatted to three decimal places.

- For this project all checks for equality on doubles should be done by checking that the values are within the range of +/– 0.0001 of each other. Example: A = 1.0133, B = 1.0134, and C = 1.0131. A and B would be considered equal, but neither A nor B would be equal to C.
- Expanded toString() examples for Triangle, Face, and Square in order:

Triangle

[A(x0.000, y0.000, z1.000); B(x1.000, y–1.000, z1.000); C(x1.000, y0.000, z1.000); N<0.000i, 0.000j, 1.000k>]

Face (Note no new line characters)

{F[A(x0.000, y0.000, z1.000); B(x1.000, y–1.000, z1.000); C(x1.000, y0.000, z1.000)] [A(x0.000, y0.000, z1.000); B(x0.000, y–1.000, z1.000); C(x1.000, y–1.000, z1.000)] N<0.000i, 0.000j, 1.000k>}

Square (Note no new line characters)

|C{F[A(x0.000, y0.000, z1.000); B(x1.000, y–1.000, z1.000); C(x1.000, y0.000, z1.000)] [A(x0.000, y0.000, z1.000); B(x0.000, y–1.000, z1.000); C(x1.000, y–1.000, z1.000)] N<–0.000i, 0.000j, 1.000k>}{F[A(x0.000, y–1.000, z1.000); B(x0.000, y0.000, z1.000); C(x0.000, y0.000, z0.000)] [A(x0.000, y–1.000, z1.000); B(x0.000, y0.000, z0.000); C(x0.000, y–1.000, z0.000)] N<–1.000i, –0.000j, –0.000k>}{F[A(x1.000, y–1.000, z1.000); B(x0.000, y–1.000, z1.000); C(x0.000, y–1.000, z0.000)] [A(x1.000, y–1.000, z1.000); B(x0.000, y–1.000, z0.000); C(x1.000, y–1.000, z0.000)] N<–0.000i, –1.000j, 0.000k>}{F[A(x1.000, y–1.000, z0.000); B(x0.000, y0.000, z0.000); C(x1.000, y0.000, z0.000)] [A(x1.000, y–1.000, z0.000); B(x0.000, y–1.000, z0.000); C(x0.000, y0.000, z0.000)] N<–0.000i, –0.000j, –1.000k>}{F[A(x1.000, y0.000, z0.000); B(x1.000, y0.000, z1.000); C(x1.000, y–1.000, z1.000)] [A(x1.000, y0.000, z0.000); B(x1.000, y–1.000, z1.000); C(x1.000, y–1.000, z0.000)] N<1.000i, 0.000j, –0.000k>}{F[A(x0.000, y0.000, z0.000); B(x0.000, y0.000, z1.000); C(x1.000, y0.000, z1.000)] [A(x0.000, y0.000, z0.000); B(x1.000, y0.000, z1.000); C(x1.000, y0.000, z0.000)] N<–0.000i, 1.000j, –0.000k>}|

Testing

We have included a `GeoFactory.java` for you to test your program. There is not a `RunLocalTest.java` provided for this assignment. The "Run" button will confirm that every required class is present and that they compile, in addition to checking style.

You are encouraged, but not required, to write your own `RunLocalTest.java` based off either the one from Project 1 or HW5. Writing your own test cases will be a part of the Team Project later in the course.

Submit

After testing your solution and verifying that it meets the requirements described in this document, you can submit on Vocareum. You have unlimited submissions but only the most recent submission will be graded. Grades will not be released until after the late due date of the project.