Lab Java – tutorial by Michał

Będę bazował na szablonie dostępnym na: http://tnij.org/java_szablony

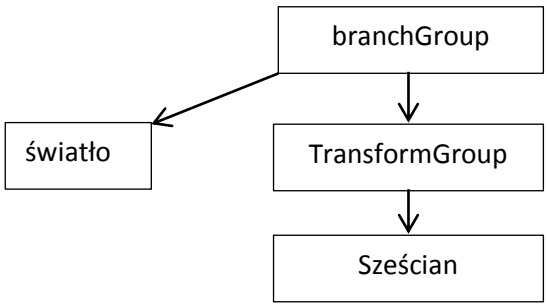
I jeszcze wskazówka odnośnie samego koła, wyczytana w tutorialu dla AiR:
„Na ostatniej labie jest koło – podobne do laborek (zrób figurę, niech się obraca w określony sposób, dodaj różne oświetlenia itp.) + parę pytań testowych."

1. Najprostszy program z obracającą się figurą

Otwieramy projekt i przechodzimy od razu do linijki:

```
BranchGroup Scena = new BranchGroup();
```

Pod nią zaczynamy pisać kod. Ale najpierw kilka słów wstępu, odnośnie samej Javy3D. Najważniejsza rzecz: opiera się o hierarchię drzewa. Jest podział na węzły/liście np.



Na tym drzewie występuje TransformGroup, na którym stosujemy przekształcenie np. zmianę pozycji czy skalowanie. To, co zastosujemy będzie miało skutek na potomkach tzn. będzie widoczne na sześcianie. Ale już na świetle nie, bo nie jest jego rodzicem.

Wracając do przykładu:

Nasza scena będzie się obracać, a to znaczy że musimy użyć pośrednika w hierarchii, jakim jest TransformGroup. Deklarujemy go i ustawiamy prawa dostępu (inaczej wysypie błąd). Na koniec łączymy węzeł z rodzicem (addChild):

```
TransformGroup TG_Glowny = new TransformGroup();
TG_Glowny.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);
Scena.addChild(TG_Glowny);
```

Teraz deklarujemy obiekt, który będzie sam z siebie obracał naszym modelem (którym będzie walec):

```
Alpha SzybkoscAnimacja = new Alpha(-1,5000);
RotationInterpolator obracacz = new RotationInterpolator(SzybkoscAnimacja, TG_Glowny);
```

W zmiennej SzybkoscAnimacja, druga wartość w nawiasie oznacza czas (w ms), przez który animacja ma się w pełni odtworzyć. Im mniejsza liczba, tym szybsza animacja.
Teraz, chcemy mieć coś, co będzie trzymało pewne obiekty w stałej pozycji np. światła. Deklarujemy więzy, ustawiamy je jako więzy interpolatora. Na koniec dodajemy już gotowy interpolator do TransformGroup:

```
BoundingSphere Wiezy = new BoundingSphere();
obracacz.setSchedulingBounds(Wiezy);
TG_Glowny.addChild(obracacz);
```

Po odpaleniu programu nic by nie było widać (czarne okno), bo nie ma żadnego źródła światła. Dodajmy je, ustawmy więzy i dodajmy bezpośrednio do sceny (**!!!**), aby się nie obracało na orbicie, jak walec:

```
AmbientLight Swiatlo = new AmbientLight();
Swiatlo.setInfluencingBounds(Wiezy);
Scena.addChild(Swiatlo);
```

Dodaliśmy światło rozmyte, tzn. jest rozproszone i miękko oświetla scenę.
W oknie nadal nic nie widać, bo nie ma obiektu 3D do obracania. Najpierw ustalmy jego wygląd:

```
Appearance  WygladCylindra = new Appearance();
WygladCylindra.setColoringAttributes(new ColoringAttributes(0.5f,0.2f,0.9f,
                                                    ColoringAttributes.NICEST));
```

W drugiej linijce, w nawiasie ustawiamy kolorowanie tzn. trzy liczby oddzielone przecinkami to składowe koloru (jak RGB). Mają na końcu literę **f**, bo mają być typu **f**loat. Minimalna liczba to 0, a maksymalna to 1f. Więc, jeśli chcemy mieć czysto-czerwony kolor, wpisujemy
1.0f, 0.0f, 0.0f, co odpowiada kolorowi RGB (255, 0, 0).
No dobra, wygląd gotowy do zastosowania. Deklarujemy, w końcu, nasz walec:

```
Cylinder walec = new Cylinder(0.1f, 0.6f,
                              WygladCylindra);
```

W nawiasie są kolejne 2 liczby. Tym razem to skalowanie. W tym przypadku ustawiamy wymiar promienia na 0.1 oraz długość walca na 0.6f.
Walec zrobiony. Ale po odpaleniu programu, obracałby się jedynie wokół własnej osi. Musimy więc go oddalić od orbity. Deklarujemy nowe przekształcenie Transform3D oraz wybieramy, że ma przenieść walec na pozycję 0.9f, 0.1f, 0.0f.

```
Transform3D  ZmianaPolozenia   = new Transform3D();
ZmianaPolozenia.set(new Vector3f(0.9f,0.1f,0.0f));
```

Deklarujemy nowy TransformGroup, ustawiamy mu zadeklarowaną transformację (aby przenosił wszystkich potomków na nową pozycję) i dodajemy walec do tej grupy, a następnie grupę do TG_Glowny.

```
TransformGroup TG_Cylinder = new TransformGroup(ZmianaPolozenia);
TG_Cylinder.addChild(walec);
TG_Glowny.addChild(TG_Cylinder);
```

Teraz, po odpaleniu, nasz fioletowy walec powinien być ustawiony pionowo i krążyć sobie po orbicie ☺

Kilka uwag:
- jeśli celem jest złożenie dwóch transformacji (np. przesunięcie i obrót), deklarujemy dwa różne Transform3D, a następnie w jednym używamy instrukcji iloczynu np.

```
Transform3D  Trans1= new Transform3D();
Trans1.set(new Vector3f(0.5f,0.0f,0.0f));

Transform3D  Trans2= new Transform3D();
Trans2.rotZ(Math.PI/2);
Trans2.mul(Trans1);
```

przez co otrzymamy jednoczesne przesunięcie i obrót o 90 stopni. Trzeba pamiętać, że transformację, którą w takim przypadku trzeba zastosować jest Trans2 (bo zawiera już iloczyn -> instrukcja **mul**), a nie Trans1 (który zawiera sam obrót).
- możemy dodawać światła różnego typu. Tu w kodzie użyłem światła typu Ambient, bo nie potrzeba innego. Ale można np. użyć światła kierunkowego DirectionalLight

---

2. Nakładanie tekstury

Celem będzie uzyskanie efektu nałożenia tekstury na prostopadłościan. Teksturą będzie mur.
Robimy to, co poprzednim razem, czyli przechodzimy do:

```
BranchGroup Scena = new BranchGroup();
```

Najpierw deklarujemy więzy:

```
BoundingSphere Wiezy = new BoundingSphere();
```

Teraz światło. Tu użyłem tego samego typu światła, co w punkcie 1. Deklarujemy, ustalamy więzy i dodajemy do sceny (bezpośrednio, nie trzeba TransformGroup, bo światło będzie w stałej pozycji i orientacji).

```
AmbientLight Swiatlo = new AmbientLight();
Swiatlo.setInfluencingBounds(Wiezy);
Scena.addChild(Swiatlo);
```

A teraz nowość. Tworzymy obiekt, który będzie przechowywał wygląd modelu.

```
Appearance WygladMur = new Appearance();
```

Deklarujemy teksturę i wpisujemy ścieżkę obrazka (który musi być pod ścieżką, inaczej program się nie skompiluje). Nową teksturę dodajemy do wyglądu:

```
Texture TeksturaMur = new TextureLoader("obrazki/murek.gif", this).getTexture();
WygladMur.setTexture(TeksturaMur);
```

Wygląd gotowy. Teraz zajmijmy się zmianą pozycji, czyli deklarujemy Transform3D i TransformGroup:

```
Transform3D Mur = new Transform3D();
Mur.setTranslation(new Vector3f(0.0f, 0.3f, 0.0f));
TransformGroup TGMur = new TransformGroup(Mur);
```

Ostatnią kwestią będzie deklaracja modelu 3D prostopadłościanu. W nawiasach wpisujemy wymiary (3 liczby float), odpowiednią flagę, która nałoży teksturę, oraz obiekt z wyglądem, który zadeklarowaliśmy:

```
Box MurModel = new Box(0.2f, 0.1f, 0.5f,
                       Box.GENERATE_TEXTURE_COORDS,
                       WygladMur);
```

Stworzyliśmy prostopadłościan o wymiarach 0.2 x 0.1 x 0.5. Na nim zostanie wygenerowana tekstura, która jest zawarta w WygladMuru.

Zostaje tylko dodać model do TransformGroup, a następnie samą grupę do sceny

```
TGMur.addChild(MurModel);
Scena.addChild(TGMur);
```

Po odpaleniu zobaczymy prostopadłościan z nałożoną teksturą muru:

---

3. Światła i ich łączenie

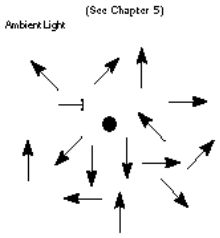
Najpierw trochę teorii. W Javie, jest wiele dostępnych typów świateł. Do tej pory był wykorzystywany jedynie Ambient light. No to po kolei:

a) Ambient light – światło tła, wszechobecne. Można powiedzieć, że jest domyślnym światłem, które pada na obiekt

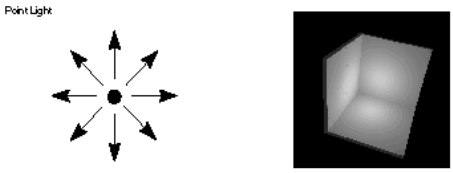
b) Directional light – ukierunkowane. Ma swój punkt położenia i rzuca równoległe promienie w danym kierunku, określonym w wektorze ukierunkowania

c) Point light – światło punktowe. Jak sama nazwa mówi, z danego punktu rzuca promienie w każdym kierunku. Takim przykładowym „point light" może być Słońce.

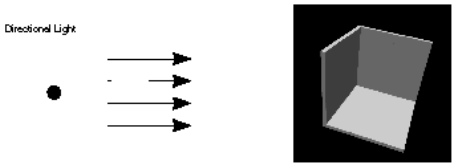
d) Spot light – światło reflektorowe. Podobne do directional, ale jest zasadnicza różnica: rzucane promienie nie są równoległe. Są raczej objęte w stożku



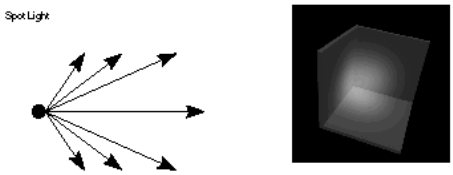
Ambient



Point



Directional



Spot

Wystarczy teorii. Czas na program, który wyświetli niebieską kulę oświetlaną światłem typu SpotLight. Jak zwykle zaczynamy od przejścia pod linię:

```
BranchGroup Scena = new BranchGroup();
```

i deklaracji węzłów (kuli ograniczającej):

```
BoundingSphere Wiezy = new BoundingSphere();
```

Pierwsze, co musimy zrobić to przygotować zmienne określające wygląd kuli. Deklarujemy materiał kuli:

```
Material material_kuli = new Material(new Color3f(0.0f, 0.1f, 0.0f),
                                      new Color3f(0.0f, 0.0f, 0.3f),
                                      new Color3f(0.6f, 0.0f, 0.0f),
                                      new Color3f(1.0f, 1.0f, 1.0f),
                                      80.0f);
```

Mamy tu aż 5 argumentów konstruktora. Po kolei:

- pierwszy argument to kolor typu **ambient**. To kolor, jaki będzie mieć obiekt w nieoświetlonym miejscu.
- drugi to kolor typu **emmisive**. Jest kolorem, który jest emitowany z obiektu.
- trzeci to kolor typu **diffuse**. To właściwy kolor materiału, który jest widoczny, gdy oświetlamy obiekt zwykłym, białym światłem.
- czwarty to kolor typu **specular**. Określa kolor odbicia światła od powierzchni obiektu.
- ostatni argument to współczynnik określający błyszczenie. Im większy, tym bardziej błyszcząca powierzchnia.

Materiał jest. Teraz deklarujemy zmienną odpowiadającą za wygląd cieniowania:

```
ColoringAttributes colAtr = new ColoringAttributes();
colAtr.setShadeModel(ColoringAttributes.SHADE_GOURAUD);
```

W drugiej linijce ustawiamy cieniowanie typu Gouraud, które dale najlepszy efekt. Jest też cieniowanie płaskie, ale wybierając je będzie widać b. wyraźnie wielokąty, z jakich zbudowany jest obiekt.

Co teraz? Łączymy to, co zadeklarowaliśmy w jedną zmienną:

```
Appearance wyglad_kuli = new Appearance();
wyglad_kuli.setMaterial(material_kuli);
wyglad_kuli.setColoringAttributes(colAtr);
```

Materiał kuli gotowy. Zastosujmy go:

```
Sphere kula = new Sphere(0.3f, Sphere.GENERATE_NORMALS, 80, wyglad_kuli);
```

Utworzyliśmy kulę (sferę) o promieniu 0,3f. Jest też argument GENERATE_NORMALS. O co chodzi? Stosując materiał, kompilator musi wiedzieć, jak jest ułożona powierzchnia obiektu. Dowiaduje się tego z NORMALS, czyli normalnych do powierzchni. W tym przypadku są ustawione na zewnątrz. Gdybyśmy zmienili ten argument na

GENERATE_NORMALS_INWARD, widzielibyśmy materiał od wewnątrz (co można zastosować przy robieni tła nieba – patrz: laborka z teksturami).

Kolejnym argumentem jest liczba całkowita, która określa z ilu wielokątów zbudować naszą kulę. Im więcej, tym ładniejsza i mniej widać wielokąty. W tym przypadku, budujemy kulę z 80 wielokątów. W ostatnim argumencie, stosujemy oczywiście wygląd, który łączy poprzednie deklaracje.

No dobra, wszystko odnośnie kuli gotowe. Czas zająć się światłami. Deklarujemy światło typu SpotLight:

```
SpotLight swiatlo_sto  = new SpotLight(new Color3f( 1.0f, 1.0f, 1.0f),
                                       new Point3f(1.5f, 1.5f, 1.5f),
                                       new Point3f(0.01f,0.01f,0.01f),
                                       new Vector3f(-1.0f, -1.0f, -1.0f),
                                       (float)Math.PI,
                                       100);
```

Podobnie, jak w przypadku deklaracji kuli, tu też jest kilka argumentów. A więc:

- pierwszy argument określa kolor światła. W naszym przypadku to światło białe.
- drugi argument określa pozycję źródła światła (reflektora/lampy)
- trzeci argument to Attenuation, o którym wspominam na ostatniej stronie. Im mniejsze wartości tego punktu, tym mocniejsze światło bliżej źródła światła. Im większe wartości, tym większy dystans, na którym zanika światło.
- czwarty argument wskazuje wektor, który wskazuje kierunek świecenia
- piąty argument to kąt stożka światła
- szósty argument jest współczynnikiem skupienia światła. Im większy, tym bardziej skupione światło

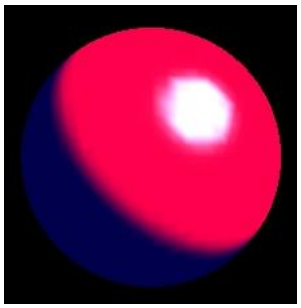
To teraz już z górki. Ustawiamy węzły światła:

```
swiatlo_sto.setInfluencingBounds(Wiezy);
```

I na koniec dodajemy światło i kulę do rodzica, jakim jest scena:

```
wezel_scena.addChild(kula);
wezel_scena.addChild(swiatlo_sto);
```

That's all. Po skompilowaniu kodu, powinno pojawić się takie coś:



Mamy kule, która jest domyślnie niebieska (składowe w DiffuseColor). Światło SpotLight jest gdzieś na górze, po prawej i świeci w kierunku kuli, co widać po rozjaśnionym kolorze Magenta. Widać biały punkt, na którym jest skoncentrowane światło. Bardziej przypomina wielokąt. Jakbyśmy zwiększyli liczbę, określającą ilość wielokątów np. do 150, ujrzelibyśmy idealną kulę.

---

4. Nałożenie nieba i manipulacja widokiem

Chyba nie musze już mówić gdzie przechodzimy ☺
Najpierw zajmiemy się niebem. Deklarujemy teksturę nieba:

```
Texture Tekstura = new TextureLoader("obrazki/clouds.gif", null, new Container()).getTexture();
```

oraz wgrywamy ją do zmiennej zawierającej wygląd:

```
Appearance Wyglad = new Appearance();
Wyglad.setTexture(Tekstura);
Wyglad.setTextureAttributes(new TextureAttributes());
```

Dobra, teraz możemy zadeklarować sferę, stosując jednocześnie zadeklarowany przed chwilą wygląd.

```
Sphere Sfera = new Sphere(6.5f,
                          Primitive.GENERATE_NORMALS_INWARD + Primitive.GENERATE_TEXTURE_COORDS,
                          Wyglad);
```

Pierwszy argument jest oczywiście promieniem sfery. Najlepiej dać jakąś wyższą wartość np. 6.5, aby nie przysłaniała wewnętrznych modeli. W drugim argumencie podajemy flagi. Traktujemy je jako liczby, więc możemy je dodawać. Pierwsza flaga oznacza, że wygląd zostanie zastosowany od środka, dzięki czemu będziemy widzieć teksturę nieba od środka kuli, a nie na zewnątrz. Druga flaga nakłada teksturę nieba na model.

Teraz przygotujmy TransformGroup, dzięki któremu będziemy mogli obracać całą sceną.

```
TransformGroup Myszka = new TransformGroup();
Myszka.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);
Scena.addChild(Myszka);
```

Dodajmy na samym środku oteksturowany walec, aby było widać obrót kamerą. Ładujemy najpierw teksturę i wyciągamy z niej obrazek

```
TextureLoader loader = new TextureLoader("obrazki/murek.gif",this);
ImageComponent2D image = loader.getImage();
```

Wgrajmy texture do zmiennej wyglądu:

```
Texture2D murek = new Texture2D(Texture.BASE_LEVEL,
                                Texture.RGBA,
                                image.getWidth(),
                                image.getHeight());

murek.setImage(0, loader.getImage());
wyglad_mury.setTexture(murek);
```

Teraz dodamy więzy BoudingSphere oraz światło. I teraz uwaga: będą one potomkami naszego TransformGroup, bo chcemy aby cała scena się obracała. Jeśli któryś element ma pozostać nieruchomy, dodajemy go bezpośrednio do sceny.

```
BoundingSphere Wiezy = new BoundingSphere();
AmbientLight lightA = new AmbientLight();
lightA.setInfluencingBounds(Wiezy);
Myszka.addChild(lightA);

DirectionalLight lightD = new DirectionalLight();
lightD.setInfluencingBounds(Wiezy);
lightD.setDirection(new Vector3f(0.0f, 0.0f, -1.0f));
lightD.setColor(new Color3f(1.0f, 1.0f, 1.0f));
Myszka.addChild(lightD);
```

Nie tłumaczę oczywiście tego fragmentu, bo to już zrobiłem w poprzednich podpunktach.

Deklarujemy walec:

```
Cylinder walec = new Cylinder(0.2f, 0.6f, Cylinder.GENERATE_TEXTURE_COORDS, wyglad_mury);
```

W argumencie możemy zobaczyć flagę, która była deklarowana przy sferze. Ta flaga też tu musi być, aby tekstura została poprawnie nałożona. Dodajemy teraz walec i sferę do TransformGroup:

```
Myszka.addChild(walec);
Myszka.addChild(Sfera);
```

A teraz gwóźdź programu. Sterowanie kamerą. Wbrew pozorom, to banalnie proste. Najpierw dajmy kontrolera obrotu:

```
MouseRotate myszObrot = new MouseRotate(Myszka);
myszObrot.setSchedulingBounds(Wiezy);
Myszka.addChild(myszObrot);
```

Po prostu zadeklarowaliśmy obiekt odpowiedniego typu, w argumencie podając TranformGroup, który będzie obracany. Pozostałe dwie linijki to dodanie do więzów i do TransformGroup.

Obrotu dokonujemy przytrzymując LPM i dokonując przesuwu.

Zostały jeszcze dwie operacje. Przesuw:

```
MouseTranslate przesMysza = new MouseTranslate(Myszka);
przesMysza.setSchedulingBounds(Wiezy);
Myszka.addChild(przesMysza);
```

, którego dokonujemy za pomocą PPM oraz przybliżanie/oddalanie:

```
MouseZoom myszZoom = new MouseZoom(Myszka);
myszZoom.setSchedulingBounds(Wiezy);
Myszka.addChild(myszZoom);
```

, którego dokonujemy za pomocą ŚPM.

To tyle.

5. Zmiana osi obrotu RotationInterpolator

To tylko krótka informacja, jak zmienić oś obrotu obracacza. Najpierw musimy ustalić oś, po jakiej ma się obracać:

```
AxisAngle4f OsObrotu = new AxisAngle4f( 0, 1, 1, (float)Math.PI ) ;
```

I teraz: pierwsze 3 argumenty to składowe. Dajemy jedynkę przy składowej, która ma być składową osi obrotu. Możemy traktować to jako określenie płaszczyzny, po jakiej będzie poruszał się obiekt. W tym przypadku mamy 0

przy osi X oraz 1 przy Y i Z, a to znaczy, że wybieramy płaszczyznę YZ, jako tą, po której ma się obracać figura. Będziemy to widzieć tak, jakbyśmy oglądali animację 2D, bo płaszczyzna jest domyślnie prostopadła do widoku.

Ostatnim argumentem jest kąt. Jeśli zależy nam tylko na zmianie osi obrotu, to wpisujemy **(float)Math.PI**. Ale możemy zmienić ten kąt, dzięki czemu obiekt będzie poruszał się, jakby po stożku.

Ładujemy naszą oś/płaszczyznę do nowego Transform3D:

```
Transform3D trObrot = new Transform3D();
trObrot.set(OsObrotu);
```

I na koniec, jeśli nie mamy zadeklarowanego obracacza, robimy to i korzystamy z metody setTransformAxis, aby ustalić ostatecznie oś:

```
RotationInterpolator obracacz = new RotationInterpolator(Szybkosc, TG_Glowny);
obracacz.setTransformAxis(trObrot);
```

Nowa oś została ustawiona.

---

6. Wzorce

Tu po prostu powstawiam szablony kodu. Zmienne:

Scena -> typ BranchGroup

Wiezy -> typ BoundingSphere

Model3D -> dowolnego typu 3D np. Box

a) dodanie światła typu Ambient do sceny

```
AmbientLight Swiatlo = new AmbientLight();
Swiatlo.setInfluencingBounds(Wiezy);
Scena.addChild(Swiatlo);
```

b) przeniesienie modelu w inne miejsce

```
Transform3D Poz = new Transform3D();
Poz.setTranslation(new Vector3f(0.0f, 0.3f, 0.0f));              //w nawiasach pozycja
TransformGroup TG_Grupa = new TransformGroup(Poz);
Grupa.addChild(Model3D);
```

c) załadowanie tekstury do wyglądu

```
Appearance Wyglad = new Appearance();
Texture Tekstura = new TextureLoader("obrazki/murek.jpg", this).getTexture();
Wyglad.setTexture(Tekstura);
```

Obiekt Wygląd możemy zastosować przy deklaracji, lub poprzez użycie metody .setAppearance()

d) łączenie przekształceń (w tym przypadku przeniesienia o 4 jednostki w górę i obrotu o 90 stopni)

```
Transform3D Poz = new Transform3D();
Poz.set(new Vector3f(0.0f,0.4f,0.0f));

Transform3D Obrot = new Transform3D();
Obrot.rotZ(Math.PI/2);

Poz.mul(Obrot);
```

e) deklaracje figur

```
Cone Stozek = new Cone(0.2f,0.3f, wygladStozka);
Cylinder Walec = new Cylinder(0.1f, 0.4f, wygladCylindra);
Box Prostopadloscian = new Box(0.2f, 0.4f, 0.3f, wygladProstopadloscianu);
```

f) tekst 3D

```
//ustawiamy czcionkę
Font3D Czcionka3D = new Font3D(new Font("Arial", Font.PLAIN, 2),
                               new FontExtrusion() );

//… i tekst
Text3D tekst3D = new Text3D( );
tekst3D.setFont3D(Czcionka3D);
tekst3D.setString("To jest tekst");

//łączymy wszystko w zmiennej Ksztalt, którą możemy dodać jako potomek do sceny/TransformGroup
Shape3D Ksztalt = new Shape3D(tekst3D, Wyglad);
```

W zmiennej Czcionka deklarujemy krój, styl i rozmiar czcionki i dodajemy efekt głębokości (dzięki czemu mamy tekst3D). A zmienna Tekst

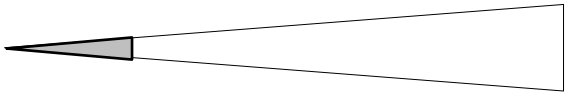
g) deklaracje świateł różnego rodzaju

```
AmbientLight swiatlo1 = new AmbientLight();
swiatlo1.setEnable(true);
swiatlo1.setColor(new Color3f( 1.0f, 1.0f, 1.0f ));    //ustawiamy kolor światła. W tym przypadku to
                                                         światło białe (po 1.0f w składowych)
DirectionalLight swiatlo2 = new DirectionalLight();
swiatlo2.setEnable(true);
swiatlo2.setColor(new Color3f( 1.0f, 1.0f, 1.0f ));
swiatlo2.setDirection(new Vector3f( 1.0f, 0.0f, 0.0f ));   //ustawiamy punkt, na który mają padać
                                                             promienie światła

PointLight swiatlo3 = new PointLight( );
swiatlo3.setEnable(true);
swiatlo3.setColor(new Color3f( 1.0f, 1.0f, 1.0f ));
swiatlo3.setPosition(new Point3f( 0.0f, 1.0f, 0.0f ));          //ustalamy pozycję źródła światła
swiatlo3.setAttenuation(new Point3f( 1.0f, 0.0f, 0.0f ));       //wyjaśnione za chwilę

SpotLight swiatlo4 = new SpotLight( );
swiatlo4.setEnable(true);
swiatlo4.setColor(new Color3f( 1.0f, 1.0f, 1.0f ));
swiatlo4.setPosition(new Point3f( 0.0f, 1.0f, 0.0f ));
swiatlo4.setAttenuation(new Point3f( 1.0f, 0.0f, 0.0f ));
swiatlo4.setDirection(new Vector3f( 1.0f, 0.0f, 0.0f ));
swiatlo4.setSpreadAngle( 0.785f );                          //ustalamy kąt stożka światła
swiatlo4.setConcentration( 0.0f );                          //skupienie promieni świetlnych
```

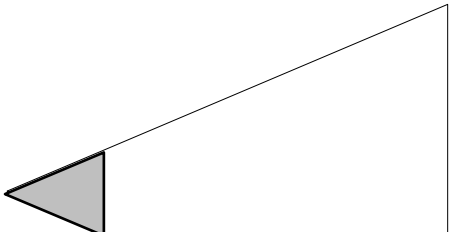
Wyjaśniam co robi **setAttenuation:**
Ta metoda określa, jak szybko światło ma zanikać w miarę oddalania się od źródła (czyli jak szybko słabnie). Metoda ta ma 3 argumenty liczbowe:
- Constant attenuation - współczynnik odpowiadający za równomierny, stały zanik światła
- Linear attenuation - ----------||---------- liniowy zanik światła
- Quadratic attenuation - ----------||---------- kwadratowy (f. kwadratowa) zanik światła

Wyjaśniam co robi **setSpreadAngle:**
W argumencie podajemy kąt stożka światła. Niżej, przedstawiam schematycznie o co chodzi, przy założeniu, że szara figura jest źródłem światłą (reflektorem/lampą) i rzuca światła na prawą stronę.

<u>mała wartość tego kąta:</u>



<u>duża wartość tego kąta:</u>



h) Timery

```
Timer czas = new Timer();                    //tworzymy Timera

TimerTask czasZad = new TimerTask(){
       public void run(){
            ...                              //wpisujemy instrukcje co cyklicznego wykonywania
       }
};

czas.schedule(czasZad, 0, 1500);             //uruchamiamy zadanie. W tym przypadku
                                             //będzie się wykonywało co 1,5 sekundy (1500 ms) i
                                             //zacznie się wykonywać z opóźnieniem 0 sekund (0 ms)
```