Final Java Projekt: 14.-18. 10.

Um die Aufgabe zu lösen, erhaltet ihr drei Bauelemente in Form von Klassen mit den folgenden Konstruktoren:

`Ellipse(int x, int y, int width, int height, String initialColor)`

`Square(int x, int y, int dimension, String initialColor)`

`Triangle(int x, int y, int w, int h, String initialColor)`

Der Null-Punkt des Koordinaten-Systems liegt links oben. Je Bauelement steht euch eine Methode zur Verfuging, die das Bauelement auf der Zeichenoberfläche zeichnet: `void draw()`

Zusätzlich erhaltet ihr eine Klasse `Canvas` die eure Zeichenoberfläche darstellt.

Weiter stehen euch folgende Farben zur Verfügung:

white	
black	
blue	
yellow	
green	
camouflageGreen	
firGreen	
magenta	
red	
gray	
brown	

Hut

Hütten bestehen aus einem Quadrat und einem aufgesetzten Dreieck. Hütten haben eine x- und y-Koordinate, sowie eine Größenangabe. Die Koordinaten referenzieren die linken obere Ecke der Grundmauer der Hütte. Die Größenangabe bezieht sich auf die Höhe und Breite der Grundmauer, sowie auf die Höhe des Daches. Die Grundmauern haben immer die Farbe brown, Dächer immer die Farbe camouflageGreen.

Implementieret ihr eine Methode `draw()`, die eine Hütte zeichnet.

Tree

Für die Kulisse soll es die Möglichkeiten geben, Bäume zu zeichnen (`draw()`). Jedem Baum wird bei der Erstellung eine x- und y-Position auf der Zeichenfläche zugewiesen. Diese referenziert den linken oberen Eckpunkt des Stammes.

Es wird zwischen Nadelbäumen und Laubbäumen unterschieden. Der Nutzer der Klasse gibt im Konstruktor an, ob es sich um einen Nadelbaum handelt oder nicht.

(a) Nadel-baum (b) Laub-baum

Beim Zeichen, erhält jeder Baum einen Stamm der Farbe brown. Dieser ist 16 Einheiten breit und 16 Einheiten hoch. Die Baumkrone wird 12 Einheiten links und 36 Einheiten oberhalb von der Position

des Stammes gezeichnet. Damit liegt der Stamm etwas „hinter“ der Baumkrone. Unabhängig von der Form ist eine Baumkrone 40 Einheiten hoch sowie breit.

Je nach Baumart unterscheidet sich Farbe und Form der Baumkrone. Nadelbäume besitzen eine spitze Baumkrone der Farbe firGreen. Laubbäume hingegen haben eine kreisförmige Baumkrone der Farbe green.

Forest

Mehrere Bäume können zu einem Wald **Forest** zusammengefasst werden. Ein Wald steht immer an einer bestimmten x- und y-Koordinate. Ihr könntet euch vorstellen, dass die Bäume auf einem Raster ähnlich einem Schachbrett gepflanzt werden. Die x- und y-Koordinate des Waldes referenziert die x- und y-Position des ersten Baumes, der am linken oberen Rand des Rasters gezeichnet wird.

Zusätzlich erhält der Wald die Anzahl der Bäume pro Spalte und Reihe.

Die Klasse **Forest** erhält zwei Methoden. Jeder dieser beide Methoden darf maximal einen Parameter erhalten.

Die Methode **drawHomogeneousForest** soll einen Wald mit nur einer Baumart zeichnen. Dabei gibt der Aufrufer an, ob es sich um einen Nadelwald handelt oder nicht (dann handelt es sich natürlich um einen Laubwald).

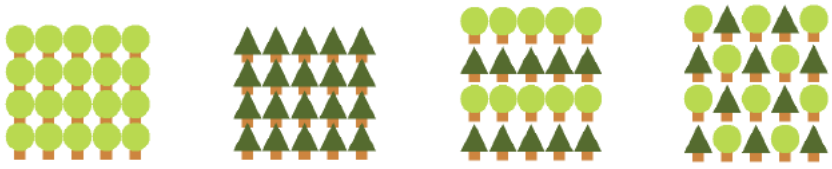
Eine zweite Methode **drawHeterogeneousForest** soll Mischwälder zeichnen bestehend aus beiden Baumarten. Der Aufrufer kann dabei angeben, ob ein Mischwald im Schachbrettmuster gezeichnet werden soll oder nicht. Letzteres bedeutet, es werden abwechselnd Reihen von Laub- und Nadelbäumen gezeichnet.

Beachten Sie bei der Implementierung folgendes:

- Besteht ein Wald aus Bäumen des gleichen Typs, so haben die Bäume auf der x-Achse einen Abstand von 40 Einheiten, auf der y-Achse einen Abstand von 45 Einheiten - gemessen an den Koordinaten des Baumes.

- Handelt es sich um einen Mischwald, so haben die Bäume auf der x-Achse einen Abstand von 40 Einheiten, auf der y-Achse einen Abstand von 55 Einheiten - gemessen an den Koordinaten des Baumes.

In folgenden Beispiel-Ausgaben findet ihr Nadelwälder und Mischwälder mit je 5 Spalten und 4 Zeilen:



(c) Laubwald (d) Nadelwald (e) Mischwald (f) Mischwald

Castle

Jedes Schloss besteht aus einer Grundmauer sowie aus Türmen und Fenstern. Die Bausteine der Grundmauer sind Quadrate. Diese können in einem von uns erdachten Raster eindeutig mittels zweier Werte *m* (Spalte) und *n* (Zeile) identifiziert werden. Der Baustein links unten beispielsweise kann mit den Werten (0, 0) eindeutig im Raster identifiziert werden. Rechts daneben liegt der Baustein (1, 0) usw. Wir bezeichnen diese Baustein-Angabe als Identifizierungstupel der Form (*m, n*):

(0,3)			(3,3)
(0,2)		(2,2)	
(0,1)	(1,1)		
(0,0)	(1,0)	(2,0)	(3,0)

Die Grundmauern eines Schlosses werden mit Hilfe von Quadraten der Kantenlänge 25 (in Bildschirmkoordinaten) erbaut. Jede Rasterposition wird dabei auf ein Quadrat abgebildet.

An bestimmten Spalten des Rasters stehen Türme. Jeder Turn besteht aus einem grauen Socket (Quadrat der Kantenlänge 25 in Bildschirmkoordinaten), aufgesetzt auf der Grundmauer an der entsprechende Rasterposition – sowie einem roten Dach in Form eines Dreiecks (Grundlinie 25 in Bildschirmkoordinaten, Höhe 25 in Bildschirmkoordinaten), aufgesetzt auf dem Sockel.

Zusätzlich haben Schlösser an bestimmten Stellen des Rasters auch Fenster der Farbe gelb (Quadrate der Kantenlänge 25 in Bildschirmkoordinaten).

Jedes Schloss hat eine x- und y-Position (Angabe in Bildschirmkoordinaten, beginnend mit (0,0) links unten), welche die linken obere Ecke des ersten Bausteins links unen (also dem Baustein an Rasterposition (0,0)) in Bildschirmkoordinaten referenziert:

➔ Schösser werden von unter nach ober aufgebaut.

Der Nutzer der Klasse kann im Konstruktor angeben, an welcher Position das Schloss stehen soll (x- und y-Werte in Bildschirmkoordinaten). Weitere Parameter des Konstruktors sind die Höhe in Anzahl der Quadrate sowie die Breite in Anzahl der Quadrate. Die Höhe bezieht sich auf die Grundmauern exklusive der Türme.

Jedes Schloss besitzt eine Methode **`draw()`** die ein Schloss nach den hier definierten Anforderungen zeichnet. Natürlich soll eure Implementierung der Methode **`draw()`** für Schlösser beliebiger Höhe und Breite funktionieren.

Türme: Ihr braucht eine Methode **`isTower()`** die prüft, ob an die Spaltenposition eines übergebenen *m*-Werts (Spaltenwert eines Identifizierungstupels) ein Turm gezeichnet werden oder nicht. Dabei gelten folgende Regeln:

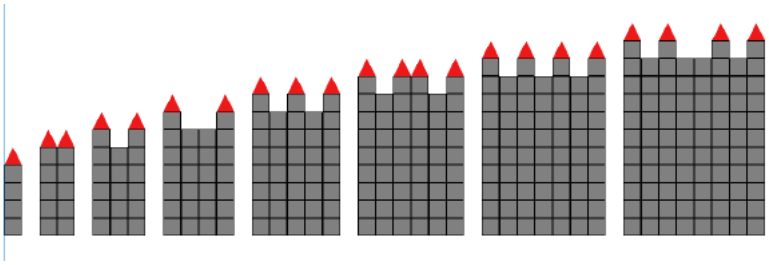
- Schlösser der Breite eins haben genau einen Turm

- Schlösser, deren Breite größer ist als eins haben mehrere Türme, die an den Rändern links und rechts beginnen

- In der Mitte dürfen maximal zwei Türme nebeneinander stehen (Siehe Schloss-Beispiel mit Breite 6)

- Würden mehr Türme nebeneinander stehen, z:B. bei Schlösser der Breite 4, so bleiben in der Mitte zwei Quadrate der Grundmauer ohne Turm

- Ansonsten haben die Türme immer einen Abstand der Breite eines Quadrats



Fenster: Schreibt dazu eine Hilfsmethode isWindow() die prüft, ob an einem übergebenen *m*- und *n*-Werte eines Identifizierungstupels ein Fenster ist oder nicht.

- Generell liegen Fenster genau an den Spaltenpositionen des Rasters, wo keine Türme sind
- Die ersten Fensterreihe beginnt immer in der zweite Reihe (der Grundmauer) von unter
- Danach folgen ggfs. Weitere Fensterreihen im Abstand von einem Quadrat
- Fensterreihen enden mindestens eine Reihe unterhalb der Türme