Aufgabenblatt 3: Typen und Vererbung

Level 1

Aufgabe 1

Ergänzen Sie Funktion `execute` aus der Klasse `Activity` (Aufgabenblatt 2) um folgende Funktionalität:

- Wenn die Aktivität „Kekse backen“ lautet, dann soll ein `Item` „Keks“ dem `inventory` des Pet hinzugefügt werden.
- Wenn die Aktivität „Laufen“ lautet, soll auf der Konsole „Puh!“ ausgegeben werden
- Wenn die Aktivität „Fußball spielen“ lautet, soll im inventory geprüft werden, ob ein Ball vorhanden ist. Nur dann soll die Aktivität wirksam sein, d.h. `engergyImpact` und `happinessImpact` werden nur dann hinzugerechnet.

Aufgabe 2

Erstellen Sie nun für die verschiedenen Aktivitäten (Kekse backen, Laufen, Fußball spielen) **jeweils eine eigene Unterklasse**. Überlegen Sie, welche Logik in die Unterkasse gehört. Passen Sie den bestehen Code entsprechend an. Was ist der Unterschied zur Implementierung in Aufgabe 1? Welche Vorteile ergeben sich?

Aufgabe 3

siehe nächste Seite

Aufgabe 3

Das Haustier soll ein paar Blumen einsammeln. Erzeugen Sie zunächst einige Blumen auf der Stage.

Nutzen Sie `Assets.flowers.FLOWER1, Assets.flowers.FLOWER2` usw.

Für den `Actor`, der das `Pet` repräsentiert, registrieren Sie anschließend einen `IntersectionEventListener`. Dieser wird von der Stage benachrichtigt, sobald es eine Überschneidung von zwei Aktoren gibt.

Hierfür müssen Sie zunächst das Interface `IntersectionEventListener` implementieren. Ein Intersection-Event tritt bei der Überschneidung (Kollision) von zwei Aktoren ein. Sobald das Haustier mit den Blumen kollidiert, sollen diese vom Bildschirm verschwinden.

Tipp: Sie können entweder eine eigene Klasse definieren, in der die Methoden des Interfaces `IntersectionEventListener` implementiert werden oder Sie definieren ein anonymes Objekt, welches das Interface implementiert.

Beispiel für Variante 1:

```
class IntersectionReaction:IntersectionEventListener {
    override fun onStartIntersection() {
        // Ihr Code
    }

    override fun onStopIntersection() {
        // Ihr Code
    }
}

yourActor.addIntersectionEventListener(
    theOtherActor,
    IntersectionReaction()
)
```

Tipp: Schauen Sie sich die Drag & Drop Beispiele in der Dokumentation an.

Aufgabe 4

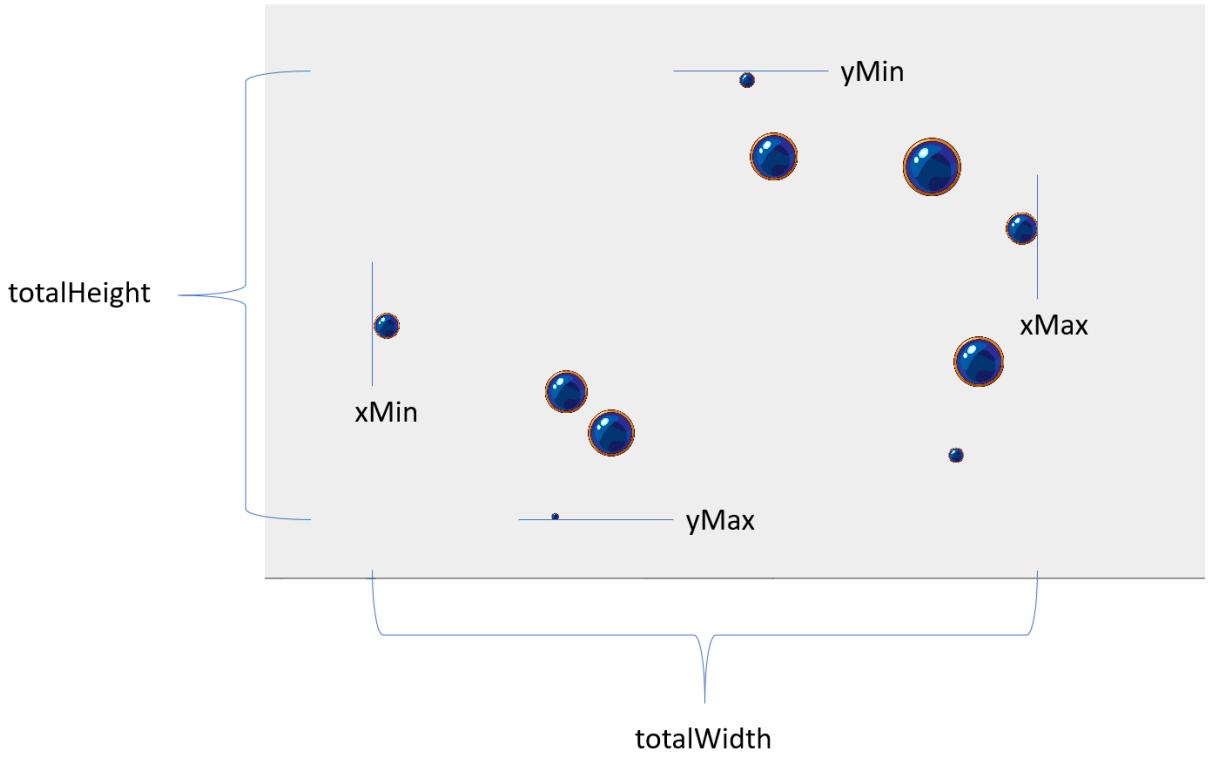
Definieren Sie eine abstrakte Klasse `StageLayout`. Diese Klasse (bzw. die konkreten Unterklassen) soll mehrere Actors nach verschiedenen Regeln auf der `Stage` arrangieren, z.B. mittig zentriert, links oder oben ausgerichtet.

Die abstrakte Klasse soll eine abstrakte Funktion enthalten.

`arrange()` → Ordnet die im `StageLayout` gespeicherten Actors an.
Die Anordnung hängt von der Unterklasse ab.

Die abstrakte Klasse soll folgende Teilimplementierung bereits enthalten:

- Speichern einer Liste von Actor-Objekten
- `xMin` liefert die kleinste x-Location aller Aktoren der Liste
- `xMax` liefert die größte x-Location aller Aktoren de Liste, dabei soll vom Actor, der am weitesten rechts positioniert ist, die x-Koordinate der rechten Seite des Actors wiedergegeben werden
- `yMin` und `yMax` liefern diese Werte entsprechend für die y-Achse
- `totalWidth` liefert die horizontale Spannbreite aller Aktoren
- `totalHeight` liefert die vertikale Spannbreite aller Aktoren



Definieren Sie folgende konkrete Unterklassen:

`AlignTop`: `arrange()` richtet alle Aktoren an `yMin` aus
`AlignLeft`: `arrange()` richtet alle Aktoren an `xMin` aus

Die Unterklassen nutzen also die Teilimplementierung der abstrakten Oberklasse.

Überlegen Sie sich eine weitere Unterklasse zum Anordnen der Aktoren.

Level 2

Aufgabe 1

Erzeugen Sie eine Unterklasse von `Actor` namens `ButtonActor`. Dieser soll das Erzeugen von Buttons vereinfachen. Hierfür bekommt der `ButtonActor` den Text des Buttons sowie eine Location übergeben und konfiguriert den Actor so vor, dass der Aktor eine gute Größe für Buttons besitzt, der Button-Text angezeigt wird und ein sinnvoller Text-Hintergrund verwendet wird, z.B. `Assets.textBackgrounds.SIMPLE_BUTTON` oder `Assets.textBackgrounds.BLUE_BUTTON`

Tipp 1: Schauen Sie sich die Text-Beispiele in der Dokumentation an.

Tipp 2: Text-Hintergründe für Buttons finden Sie in `Assets.textBackgrounds.*`

Aufgabe 2

Erzeugen Sie eine Unterklasse von `Actor` namens `Countdown`. Hier soll der spezialisierte `Actor` so vorkonfiguriert werden, dass er einen Countdown darstellt. Die Unterklasse bekommt die beiden Eigenschaften `duration` (z.B. 10 Sek) und `progress` (verstrichene Zeit) übergeben. Verwenden Sie den `reactionForTimePassed` Codeblock, um auf das Verstreichen einer Zeiteinheit zu reagieren (ähnlich wie bei `lifeGoesOn`).

Implementieren Sie zum Zeichnen das `Drawable`-Interface:

```
interface Drawable {

    fun draw ( pen: Pen )

}
```

Objekte, die diesem Interface entsprechen, können der `canvas`-Eigenschaft eines `Actors` zugewiesen werden.

Beim Zeichnen eines `Actors` auf der Stage wird am Ende vom System die `draw(…)` Methode aufgerufen. Diese erhält über den Parameter `Pen` die Möglichkeit Linien, Rechtecke und Ovale zu zeichnen.

Tipp: Schauen Sie sich die **Drawable** – Beispiele in der Dokumentation an.

Aufgabe 3

Die veränderten Werte von `happiness` des `Pet` sollen als Liniendiagramm gezeichnet werden. Erzeugen Sie hierfür eine Klasse namens `Lifeline`, die das `Drawable` Interface implementiert.

Die Klasse muss über Änderungen der `happiness` informiert werden. Es sollen die letzten `happiness` Werte gespeichert und gezeichnet werden.

Level 3

Aufgabe 1

Erzeugen Sie eine Unterklasse von `Actor` namens `DropActor`. Dieser erzeugt zunächst eine Fläche, auf die Aktoren "fallen gelassen" werden können (Drop-Area). Wenn ein Aktor auf dieser Fläche fallen gelassen wird, wird dieser innerhalb der Fläche automatisch zentriert. Wenn ein weiterer Aktor auf dieser Fläche fallen gelassen wird, werden beide Aktoren zentriert nebeneinander angeordnet.

Zur Umsetzung muss den Aktoren, die verschoben werden können, ein `DropEventListener` zugefügt werden.

Tipp: Schauen Sie sich die Drag&Drop-Beispiele in der Dokumentation an.

Aufgabe 2

Erzeugen Sie eine Unterklasse von `DropActor` namens `RemainingSpaceDropActor`. Dieser erweitert das Verhalten vom `DropActor`, indem angezeigt wird, wie viel horizontaler Platz noch verfügbar ist.

Aufgabe 3

Erzeugen Sie eine Unterklasse von `DropActor` namens `PermittedDropActor`. Dieser limitiert das Verhalten vom `DropActor`, indem nur bestimmte Aktoren auf die Fläche fallen gelassen werden können. Diese bestimmten Aktoren werden der Klasse als Parameter übergeben.