Dieses Beispiel baut auf dem Kapitel
Prototyping auf!

Buzzing bees

!

In diesem Beispiel soll eine Bienenkolonie implementiert werden. Dazu bedienen wir uns der prototypischen Vererbung; das Beispiel ist somit geeignet für Jahrgangsstufen 9/10. Es wird wenig Vorwissen benötigt; dennoch wäre es gut, wenn die Zielgruppe bereits erste Erfahrungen mit blockbasierten Sprachen, z. B. in Form von Scratch gesammelt hat. Besteht bereits Vorwissen zu UML, so können Schülerinnen und Schüler ihr vorhandenes Wissen auf ein neues Feld übertragen.Wir verwenden neue Bedienkonzepte, die blockbasierte Sprachen uns ermöglichen - wie etwa **Tinkering** und **Direct Drive**. Explorativ wird hier **Prototyping,** ein wichtiges Konzept blockbasierter Programmierung, erforscht.

Das Material besteht aus **Schülermaterialien** und **zusätzlichen Anmerkungen für Lehrkräfte** zur Umsetzung im Unterricht sowie **weiterführenden Erklärungen** und **Lösungen.**

Somit ist es möglich, das Material sowohl für die **eigene Weiterbildung** als auch in Auszügen für **Arbeitsblätter für den Schuleinsatz** zu verwenden.

Überblick über die verwendeten Symbole in diesem Handout:

	Aufgabe
	Bonusaufgabe für Fortgeschrittene, bzw. Benutzer mit Vorerfahrung
!	**Hinweis**
	Tinkering: hier gibt es keine falschen Antworten!
	Gespräch: Reflektion, Diskussion oder Austausch mit dem Banknachbarn
	Tipp: Hinweise, bzw. Ideen für die Umsetzung im Unterricht

Buzzing bees

In diesem Modul simulieren wir das Verhalten von Bienen!

Vorgeschmack auf das fertige Projekt: bit.ly/snap-bees-solution

Link zum Projekt: bit.ly/snap-bees

Öffne das Projekt über den Link aus der Box.

“Always hire a lazy person to do a difficult job - because a lazy person will find an easy way to do it."
- Bill Gates, Microsoft

Programmierer sind faul. Daher suchen sie immer nach einfachen Lösungen.
Ein Beispiel dafür ist das Programmierkonzept der **Vererbung, bzw. des Prototyping.**

Anhand eines Beispiels aus der Biologie werden wir lernen, wie uns Prototypen, bzw. die prototypische Vererbung dabei helfen kann, komplexe Sachverhalte simpel zu lösen.

Zunächst sollten wir uns im Klaren sein, was wir genau implementieren möchten:

*Es soll **drei** Arten von Bienen geben, die unterschiedliche Tätigkeiten in der Bienenkolonie ausführen.*
***Jede** dieser Arten besitzt eine **bestimmte Größe, X-und Y-Koordinaten,** ein **Kostüm,** ein **Summgeräusch,** einen **Drehtyp** (bestimmt, wie das Objekt sich auf der Leinwand dreht), und einen **Namen**. Außerdem **summen** sie.*

Die Bienenarten unterscheiden sich folgendermaßen:

- ***Sammlerinnen** sind etwas **kleinere Bienen.** Ihre Aufgabe ist es, **Honig zu sammeln.***
- ***Drohnen** sind **ebenfalls etwas kleinere Bienen, sind aber größer als Sammlerinnen.** Sie **summen auch, aber tiefer als normale Bienen**. Außerdem **verteidigen sie die Kolonie.***
- ***Königinnen** sind **größer als Drohnen. Sie summen nicht.** Ihre Aufgabe ist die **Erzeugung neuer Bienen.***

Wie modellieren wir das obige Szenario also in Snap!? Zur Veranschaulichung modellieren wir die Kolonie in **UML** und gehen dabei wie in **prototypischer Vererbung üblich vom Konkreten zum Abstrakten,** indem wir **Unterschiede** zwischen den einzelnen Objekten identifizieren.

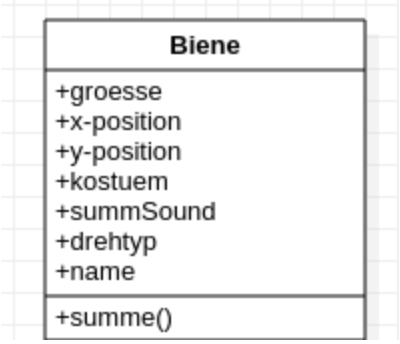
Aufgabe: Modelliere das **Bienen-Objekt in UML.**

Dieses Objekt können wir nun **in Snap implementieren!**

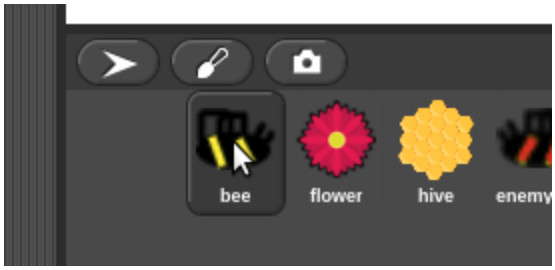
Tipps:
Unser Ziel ist es, eine Bienenkolonie zu implementieren. Natürlich besteht eine solche Kolonie besteht nicht nur aus einer einzigen Biene; sie dient uns jedoch als **Prototyp**. Indem wir diesen Prototyp **klonen**, können wir Objekte erzeugen, die von ihm **erben.**

In **Snap!** arbeiten wir nicht mit abstrakten Klassen. Stattdessen beschreiben wir konkret die **Objekte**. Das bedeutet, dass wir Prototypen und Klone identifizieren. Klone erben standardmäßig alles von ihren zugehörigen Prototypen, und können manches davon überschreiben.

Lösung für das UML-Diagramm des Bienen-Objekts:



Nachdem wir das Projekt öffnen, sehen wir einige Objekte, sog. **Sprites**.



- Wähle das **bee-Objekt** aus.
- Uns interessiert momentan **nur** dieses Objekt - du musst in den anderen Objekten **noch nichts** tun!
- **Keine Objekte löschen:** Wenn Objekte gelöscht werden, musst du **wieder von vorne anfangen, da dieser Schritt NICHT rückgängig gemacht werden kann!**

Im **Skriptbereich** (große graue Fläche links) bauen wir Blöcke aus der **Blockpalette** zu Skripten zusammen und **führen sie aus, indem wir sie anklicken.**

Da sich dieses Beispiel auf das Konzept der **Vererbung** stützt, ist es an dieser Stelle hilfreich, sich bewusst zu machen, welche Eigenschaften das Objekt besitzt, bzw. welche es vererben **kann.**

Aufgabe: Durchstöbere das Objekt **bee**.
Notiere, welche **Eigenschaften/Merkmale** das Objekt besitzt.

Blöcke(Skriptbereich)_ _____ _____

_____x-Position_____ _____ _____

Wenn ein Objekt geklont wird - und das werden wir mit der Biene tun - dann werden **alle diese Eigenschaften, bzw. Merkmale an die Klone vererbt.**

Da der **Skriptbereich**, also die verwendeten Blöcke dazu gehören, bedeutet das, dass ein Objekt seinen **Skriptbereich vererben** kann.
Die Klone zeigen dann dasselbe Verhalten wie der Prototyp.

Bisher verfügt unsere Biene nur über **einen einzigen Block** - nämlich 

.

Aufgabe: Baue für die Biene ein Skript, das gestartet werden soll, wenn die grüne Flagge geklickt wird. **Das Skript soll bewirken, dass der** 

-Block in einer Endlosschleife ausgeführt wird.

Lösungen:

- Blöcke (Skriptbereich)
- X-Position
- Y-Position
- Klänge
- Kostüme

Weniger wahrscheinlich:

- Größe
- Grafikeffekte
- Richtung
- Art der Drehung (rotierend/nur seitlich/überhaupt nicht)
- Klickbar?

Skript zum Summen:

Tipps:
Da **Prototyping** auf **Standardobjekten (= Prototypen)** und **Abweichungen von diesen** basiert, ist das Identifizieren von Eigenschaften und Merkmalen ein zentraler Schritt.
Daher ist es wichtig, dass sich Schülerinnen und Schüler bewusst machen, welche Eigenschaften Objekte in Snap! überhaupt besitzen - im echten Leben geschieht dieser Schritt nämlich nur selten explizit.
Die folgende Übung wird das demonstrieren.

Im nächsten Schritt wenden wir uns nun den Sammler-Bienen zu. Wie im obigen Textabschnitt erkennbar, beschreiben wir **ähnliche** Objekte typischerweise ausgehend von ihren **Unterschieden** zueinander. **Dieses Konzept nennt sich Prototyping.**

Aber wie funktioniert Prototyping?

1. Dein Lehrer, bzw. deine Lehrerin zeigt dir ein **Objekt.**
   Dieses Objekt heißt "Bob".
2. Einer deiner Mitschüler soll **Bob malen,**
   steht jedoch an der Tafel und sieht ihn somit nicht!
   Deine Aufgabe ist es, Anweisungen zu geben, wie Bob zu malen ist.

Aufgabe: Notiere, was du **beschrieben hast!**
(Tipp: Welche Anweisungen hast du deinem Mitschüler gegeben?)

_____

_____

_____

1. Dein Lehrer, bzw. deine Lehrerin zeigt dir nun ein **weiteres Objekt.**
   Dieses Objekt heißt "Bill".
2. Dein Mitschüler soll nun **Bill malen,** und zwar **direkt neben Bob.**
   Wie vorher steht der Mitschüler jedoch an der Tafel **und sieht ihn somit nicht!**
   Deine Aufgabe ist es, Anweisungen zu geben, wie Bill zu malen ist.

Vermutlich war die Beschreibung von **Bill** deutlich **simpler** als die Beschreibung von **Bob!**

Aufgabe: Notiere, was du **diesmal beschrieben hast!**
(Tipp: Welche Anweisungen hast du deinem Mitschüler gegeben?)

_____

Tipps:
Grundsätzlich basiert diese Übung auf dem Elefanten-Beispiel, das oftmals mit Prototyping in Verbindung gebracht wird:
***"Tim hat noch nie einen Elefanten gesehen**. Dann sieht er den grauen Elefanten "Fred". **Da Fred der erste Elefant ist, den Tim je gesehen hat, ist Fred für Tim der "Prototyp",** d.h. er repräsentiert in diesem Moment alles, was Tim unter dem Begriff "Elefant" versteht.*

Das bedeutet auch, dass Fred für Tim alle Eigenschaften verkörpert, die einen Elefanten ausmachen.
Dann sieht Tim einen weiteren Elefanten - "Clyde" (s. links). ***Wie würde Tim diesen Elefanten wohl beschreiben?*** *Vermutlich würde er sagen* ***"Er ist wie Fred, nur weiß!"***
Es scheint intuitiv, *mithilfe von Prototypen und Abweichungen von Prototypen neue Objekte in unserer Lebenswelt zu erklären. Diese Lerntheorie finden wir auch im Bereich des Spracherwerbs."*

Um dieses Szenario auf das Klassenzimmer zu übertragen, werden die beiden Figuren **Bill** (bit.ly/snap-bill) und **Bob** (bit.ly/snap-bob) benötigt.
Eine Schülerin, bzw. ein Schüler erhält diverse Kreiden/farbige Stifte und soll an die Tafel gehen, um **Bob** zu malen - und zwar ausgehend von den Beschreibungen, die von den Mitschülern gegeben werden.

Dem Rest der Klasse wird **Bob** gezeigt. Er soll dem Maler beschrieben werden. Hier identifizieren die Schülerinnen und Schüler implizit Eigenschaften und Merkmale der Figur, und beschreiben sie. Falls die Farbe von **Bob** nicht angesprochen wird, sollte darauf hingewiesen werden.

Im zweiten Schritt wird dem Maler an der Tafel dann **Bill** beschrieben. Hier ist zu erwarten, dass die Schülerinnen und Schüler **der Einfachheit halber** sagen werden "Wie Bob, nur grün.". Tun sie das, so haben sie unbewusst und intuitiv das **Prototyping-Konzept verwendet: Sie haben Merkmale und Unterschiede zwischen Objekten identifiziert, Bob als Prototypen verwendet und Bill über seine Abweichung (hier: Farbe) von diesem Prototypen definiert.**

Wir benutzen nun **Prototyping,** um die Sammler-Biene ausgehend von ihren Unterschieden zu einer “Standard”-Biene zu definieren.
Die Standardbiene haben wir oben bereits modelliert!

Aufgabe: Notiere, welche Eigenschaften eine **Sammler-Biene** von einer **“Standard”-Biene unterscheiden.** Die Sammlerin ist “wie der Prototyp”, nur ...

_____

_____

Aufgabe: Modelliere das **Sammler-Bienen-Objekt in UML.**

Dieses Modell setzen wir nun in Snap um.

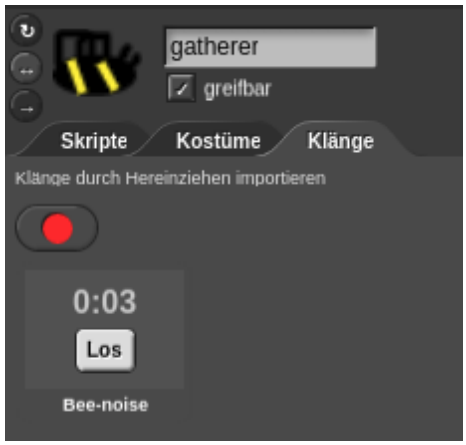
Aufgaben:

- Erzeuge einen **Klon** des **Prototypen-Objekts “bee”!** *(Tipp: siehe rechts!)*
- Wähle das neu erzeugte Objekt mit dem Namen **“bee(2)” aus!**
- Gib dem neuen Objekt einen **passenderen Namen!**

Wir überprüfen, was der Klon geerbt hat.
Dazu untersuchen wir die drei Registerkarten **Skripte, Kostüme** und **Klänge:**

Wir sehen, dass die Sammler-Biene das **Kostüm** und das **Summgeräusch** vom Prototypen **geerbt** hat.

Wir erinnern uns:
Prototyping funktioniert nach dem Prinzip **“Wie der Prototyp, nur …”**



Die Sammlerbiene ist nun also genau “wie die Standardbiene”!

Lösung:
Zur Lösung dieser Aufgabe erinnern wir uns einfach an den entsprechenden Absatz am Anfang des Moduls:

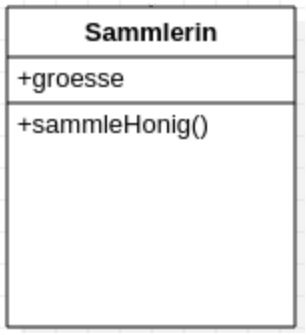
"***Sammlerinnen*** *sind etwas* ***kleinere Bienen.*** *Ihre Aufgabe ist es,* ***Honig zu sammeln."***

Diesen Absatz zerlegen wir nun in die relevanten Bestandteile.

Dabei erinnern wir uns daran, dass hier, gemäß dem Prototyping-Paradigma, Objekte ausgehend von ihren Abweichungen zum Prototypen definiert werden.

Sammlerinnen *sind etwas* ***kleinere Bienen.***
→ bedeutet für uns, dass alle Eigenschaften der **Standardbiene übernommen werden, Sammlerinnen jedoch kleiner sind, d.h.** ihre **Größe** geringer ist. Daher müssen wir die Größe im Diagramm explizit angeben, denn sie wird nicht vom Prototypen **übernommen**, sondern **überschrieben.**

Ihre Aufgabe ist es, ***Honig zu sammeln.***
→ Diese Aufgabe haben **Standardbienen** nicht, d.h. auch diese Eigenschaft muss im entsprechenden Diagramm festgehalten werden.



… oder?

Auch der Code des Prototypen wurde geerbt. Das sehen wir **einerseits** dadurch, dass das Skript, das wir dem Bienen-Prototyp vorher gegeben haben, nun hier auftaucht; wir können uns jedoch auch vergewissern, indem wir einen **Rechtsklick** auf den Skriptbereich machen.

Dann sollte das folgende Menü sichtbar werden, **in dem wir sehen, dass die Dialogbox "geerbt" mit einem Haken versehen ist.**

Um zu überprüfen, von **welchem** Objekt geerbt wird, können wir den Mauszeiger auf das Objekt halten, ohne dabei zu klicken:

Prototyping funktioniert nach dem Prinzip **"wie der Prototyp, nur …"**

...und um den "nur …"-Teil,
d.h. die **Unterschiede zum Prototypen** kümmern wir uns jetzt.

Unterschied 1: Eine Sammler-Biene soll etwas **kleiner** sein als eine "normale" Biene.

Aufgabe: Füge im Objekt "gatherer" entsprechenden Code hinzu, mit dem die Sammler-Biene **kleiner** wird als der Prototyp, von dem sie geklont wurde.

Die Lösung ist simpel. Was ist zu tun?

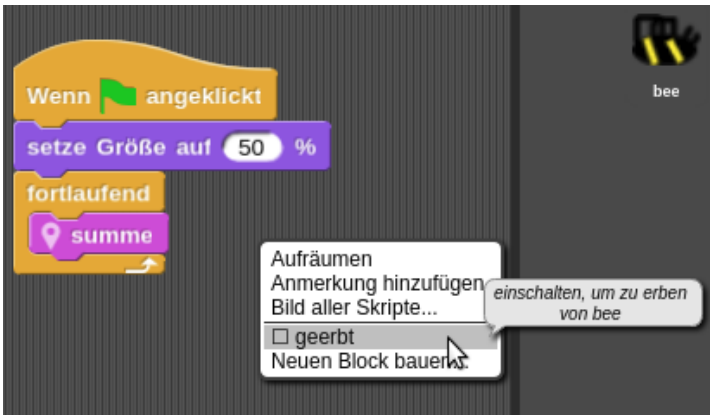
- Die Sammler-Biene erbt standardmäßig ihre **Größe** von ihrem Prototypen
- Diese Größe überschreibt sie, indem sie bei Programmstart den entsprechenden Attributwert auf 50% setzt

Lösung:
Es muss lediglich ein zusätzlicher Block eingefügt werden:

Wir haben durch diesen Eingriff den Code der Sammler-Biene geändert.

Er weicht nun von dem des Prototypen ab.

Snap! reagiert auf diese Änderung damit, dass der Skriptbereich (= der Code/das Verhalten des Objekts) **nicht mehr als geerbt markiert wird!**



Wir machen eine weitere wichtige Beobachtung.

Prototypische Vererbung ist dynamisch. Anfangs wurde das Verhalten des Klons durch das Verhalten des Prototyps bestimmt. Nehmen wir jedoch eine Änderung im Klon vor, so wird die Vererbungsbeziehung **dynamisch** angepasst.

Aufgabe: Überprüfe, ob es möglich ist, diese Anpassung wieder **rückgängig** zu machen!
Wie können wir die Änderungen **verwerfen** und wieder zu dem **geerbten Verhalten zurückkehren?**

Unterschied 2: Eine Sammler-Biene soll Pollen von Blumen sammeln.

Aufgabe: Implementiere folgendes Skript in der Sammler-Biene:

- Wenn die grüne Flagge angeklickt wurde,
- soll die Biene in einer Dauerschleife
- auf die Blume zeigen,
- sich solange vorwärts bewegen, bis sie die Blume berührt,
- auf den Bienenstock zeigen,
- sich solange vorwärts bewegen, bis sie den Bienenstock berührt

Bonusaufgabe: Lässt sich das Flugverhalten der Biene auch zufälliger, bzw. realistischer gestalten?

Lösung:
Diese Anpassung kann auch wieder **rückgängig gemacht werden**, indem wir angeben, dass der Skriptbereich geerbt werden soll. Um das zu tun, setzen wir den Haken bei “geerbt”.

Lösung für das “Honig-Sammel-Skript”:

Nun gehen wir für die Drohne analog vor. Zunächst modellieren wir ihre **Eigenschaften** und ihr **Verhalten**, und implementieren dann beides in Snap!.

Aufgabe: Notiere, welche Eigenschaften eine **Drohne** von einer “normalen” Biene unterscheiden! Die Drohne ist “wie der Prototyp”, nur ...

_____

_____

Aufgabe: Modelliere die **Drohnen-Biene in UML.**

Aufgabe:

- Erzeuge in Snap ein neues Objekt, indem du das **bee**-Objekt erneut klonst!
- Gib dem Objekt einen sprechenden Namen (etwa **drone**)
- Gib dem neuen Objekt Code, mit dem die neue Biene kleiner wird als der Prototyp, von dem sie geklont wurde.

Die Aufgabe von Drohnen ist es, die Bienenkolonie zu verteidigen.
Den entsprechenden Code implementieren wir in zwei Phasen:

Aufgabe: Erzeuge das folgende **neue** Skript im Drohnen-Objekt:

- Wenn die grüne flagge angeklickt wurde,
- soll die Biene in einer Dauerschleife
- über eine Zeitspanne von 0,3 bis 1,0 Sekunden zu einer zufälligen Position zwischen [-140, -160] [20, 40] gleiten

*Tipp: Den **gleite-**Block mit dem Zufallszahlen-Operator-Block kombinieren!*

Nun schwebt die Drohne am Eingang zum Bienenstock und ist **bereit**, ihn zu **verteidigen**.

Aufgabe: Erweitere das soeben erzeugte Skript:

- Falls das Objekt **enemy_bee** näher als 200 kommt,
- soll die Biene ein Ausrufezeichen für 0,3 Sekunden anzeigen,
- auf das Objekt **enemy_bee** zeigen,
- 20-mal folgendes wiederholen: sich 10 Schritte vorwärts bewegen

Nun bemerkt die Drohne, wenn eine fremde Biene zu nahe an den Bienenstock gerät, und reagiert darauf, indem Sie sich auf sie stürzt.

Lösung:
Zur Lösung dieser Aufgabe erinnern wir uns einfach an den entsprechenden Absatz am Anfang des Moduls:

"***Drohnen*** *sind* ***ebenfalls etwas kleinere Bienen, sind aber größer als Sammlerinnen.*** *Sie* ***summen auch, aber tiefer als normale Bienen****. Außerdem* ***verteidigen sie die Kolonie."***

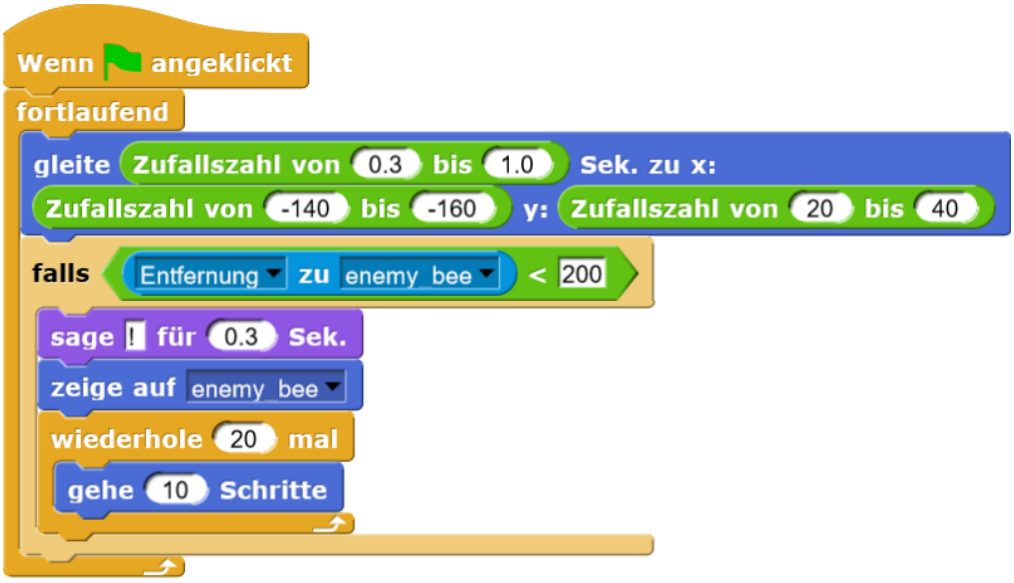
Diesen Absatz zerlegen wir nun in die relevanten Bestandteile.

Dabei erinnern wir uns daran, dass hier, gemäß dem Prototyping-Paradigma, Objekte ausgehend von ihren Abweichungen zum Prototypen definiert werden.

Drohnen *sind* ***ebenfalls etwas kleinere Bienen, sind aber größer als Sammlerinnen.***
→ Wie bei der Sammlerinnen-Biene müssen wir hier erneut die Größe überschreiben

Sie ***summen auch, aber tiefer als normale Bienen****.*
→ Wir müssen das **Summen** überschreiben

Außerdem ***verteidigen sie die Kolonie.***
→ Dieses Verhalten ist neu und tritt nur in der Drohne auf.



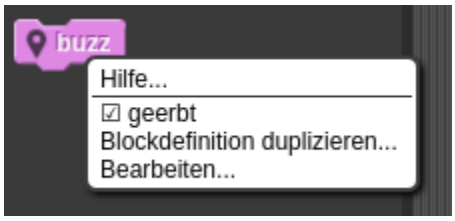
Bonusaufgabe: Erweitere das Skript ein **letztes Mal:**

- Falls das Objekt **enemy_bee** berührt wird,
- **starte** folgendes Skript:
  - lasse **enemy_bee** folgendes tun:
    - verstecke dich
    - verlasse die Leinwand
    - warte eine bestimmte Zeit
    - zeige dich wieder

Wie eine normale Biene soll auch eine Drohne summen.
Ihr Summen soll jedoch tiefer sein als das einer normalen Biene. Dazu müssen wir das geerbte Verhalten überschreiben!

Aufgabe: Finde den Block, der für das Summgeräusch verantwortlich ist, **in der Blockpalette (nicht im Skriptbereich)!**

Wir stellen fest, dass er, anders als die anderen Blöcke, die wir vorfinden, leicht ausgeblasst ist. **Dieser Effekt bedeutet, dass der Block geerbt ist. Das können wir durch Rechtsklick überprüfen.**

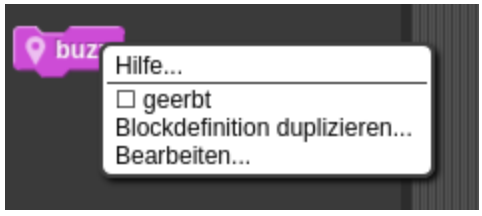


Was bedeutet das?
Es bedeutet, dass der Block - und sein gesamter Inhalt - vom Prototypen, d.h. der Standardbiene übernommen wurden **(Wir erinnern uns: "Wie der Prototyp, nur …").**

Um den Block zu überschreiben, müssen wir ihn nur bearbeiten.

Aufgabe: Verändere den Code dieses Blocks, sodass die Drone **tiefer summt** als eine normale Biene!

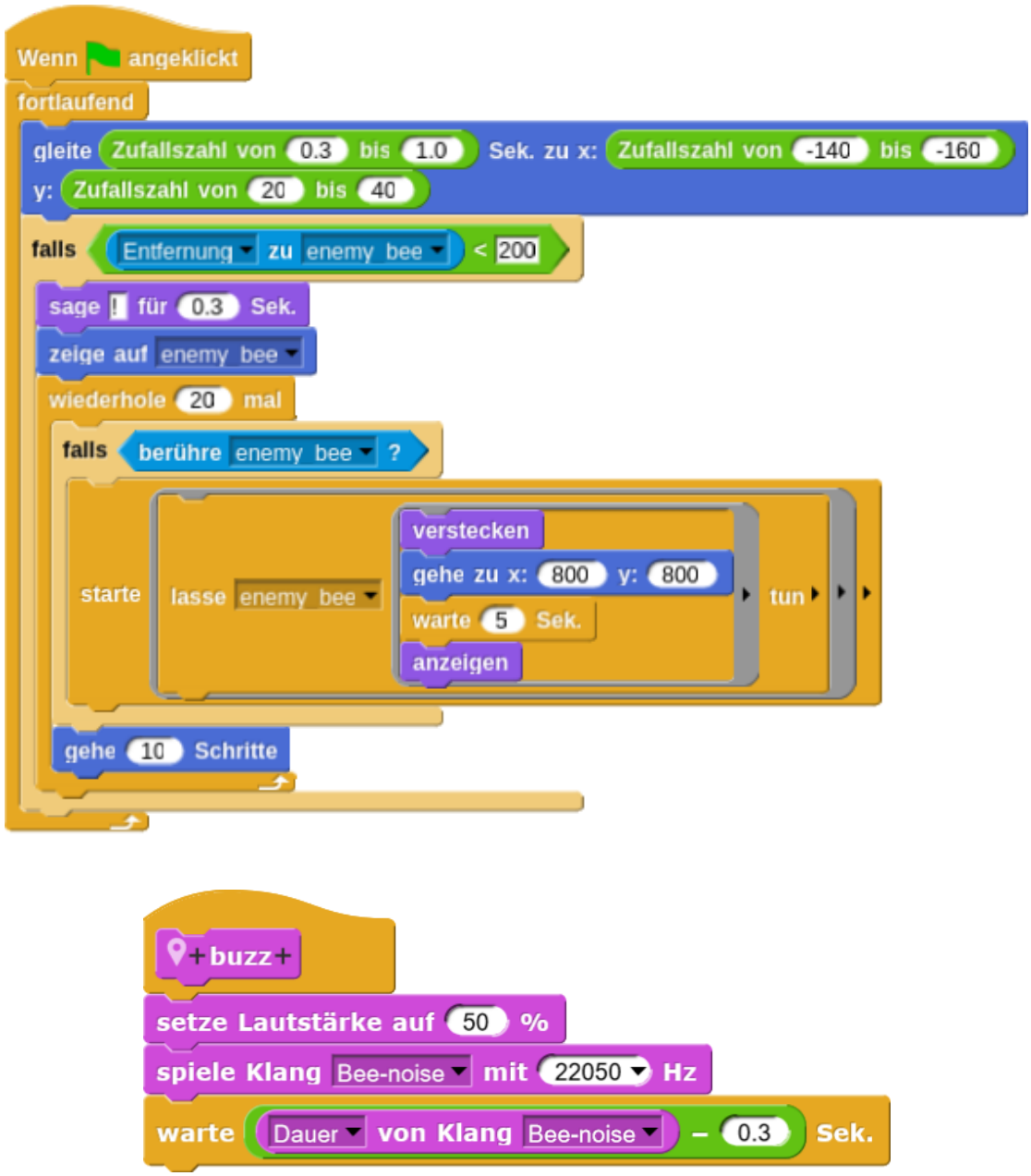
Klicke auf **OK**, bzw. **anwenden**. Wir sehen: der Block ist nicht mehr ausgeblasst und dementsprechend nicht mehr als "geerbt" markiert.



Das liegt daran, dass die Drohne den geerbten Block nun nicht mehr vom Prototypen übernimmt, sondern vom Prototypen **abweicht**.

Mit anderen Worten:
Die Drohne ist jetzt "wie der Prototyp",
nur **ist sie kleiner**, **summt tiefer**, und **verteidigt die Kolonie.**

Lösung:


Wenn angeklickt
fortlaufend
gleite Zufallszahl von 0.3 bis 1.0 Sek. zu x: Zufallszahl von -140 bis -160
y: Zufallszahl von 20 bis 40
falls Entfernung zu enemy bee < 200
sage ! für 0.3 Sek.
zeige auf enemy bee
wiederhole 20 mal
falls berühre enemy bee ?
starte
lasse enemy bee
verstecken
gehe zu x: 800 y: 800
warte 5 Sek.
anzeigen
tun
gehe 10 Schritte
+buzz+
setze Lautstärke auf 50 %
spiele Klang Bee-noise mit 22050 Hz
warte Dauer von Klang Bee-noise – 0.3 Sek.

Wir fassen bis hierhin zusammen:

Standardmäßig werden alle Eigenschaften vom **Prototypen** geerbt. Dazu gehören beispielsweise Kostüme, Klänge, Variablen (hier nicht thematisiert), oder auch Code (bzw. das Verhalten).

Möchten wir diese Eigenschaften mit eigenen Werten des Klons **überschreiben**, so können wir sie einfach im Klon **verändern** (wie beispielsweise mit der Größe, oder dem Summgeräusch). Dadurch wird die Vererbungsbeziehung **dynamisch** angepasst. Dieser Schritt kann auch wieder **rückgängig** gemacht werden, was unsere Änderungen verwirft.

Unsere Bienenkolonie ist soweit vollständig.

Aufgabe: Notiere, was die einzelnen **Bienenarten** in Snap tun!

Sammlerin: _____

_____

Drohne: _____

_____

Nun fehlen nur noch einige kleine Details, um die wir uns kümmern müssen:

- Die letzte Bienen-Art - die Königin - fehlt bislang noch komplett
- Es gibt nur eine Biene pro Art (d.h. **eine** Drohne, **eine** Sammlerin)
- Die Prototypen-Biene liegt teilnahmslos auf der Leinwand und **tut nichts**

Im letzten Schritt werden wir uns um diese Details kümmern.

Aufgabe: Notiere, welche Eigenschaften eine **Königin** von einer “normalen” Biene unterscheiden! Die Königin ist “wie der Prototyp”, nur ...

_____

_____

Aufgabe: Modelliere die **Königin in UML.**

Lösung:
Die erste Aufgabe ist reine Wiederholung. Sie kann auch einfach im Plenum besprochen werden.

Zur Lösung dieser Aufgabe erinnern wir uns einfach an den entsprechenden Absatz am Anfang des Moduls:

Königinnen** sind **größer als Drohnen. Sie summen nicht.** Ihre Aufgabe ist die **Erzeugung neuer Bienen.”

Diesen Absatz zerlegen wir nun in die relevanten Bestandteile.

Dabei erinnern wir uns daran, dass hier, gemäß dem Prototyping-Paradigma, Objekte ausgehend von ihren Abweichungen zum Prototypen definiert werden.

Königinnen** sind **größer als Drohnen.
→ Wir müssen erneut die Größe mit ins Diagramm aufnehmen, denn sie wird wieder überschrieben.

Sie summen nicht.
→ Unterschied zum Prototypen. Auch das Summen nehmen wir ins Diagramm auf.

*Ihre Aufgabe ist die **Erzeugung neuer Bienen.***
→ Eine neue Aufgabe = Abweichung vom Prototypen = ins Diagramm aufnehmen.

Tipps:
Spätestens nach diesem Durchgang dürfte den Schülerinnen und Schülern klar geworden sein, wie Prototyping funktioniert. Es werden Unterschiede zum Prototypen identifiziert, und anhand dieser Unterschiede wird dann das neue Objekt beschrieben.

Besonders günstig ist, dass die Implementierung in Snap! demselben Muster folgt.

Um die Königin zu implementieren, erzeugen wir einen **weiteren Klon der Prototypen-Biene.**

Aufgabe:

- Erzeuge ein neues Objekt, indem du das **bee**-Objekt erneut klonst!
- Gib dem Objekt einen sprechenden Namen (etwa **queen**)
- Gib dem neuen Objekt Code, mit dem die neue Biene größer wird als der Prototyp, von dem sie geklont wurde.

Die Aufgabe einer Königin ist die Erzeugung neuer Bienen. Dazu soll sie in einer Endlosschleife **neue Objekte** erzeugen - das funktioniert in Snap!, indem **geklont** wird.

… Doch was soll konkret geklont werden?

Objekte können untereinander nicht nur mit hilfe von Nachrichten (**broadcast**) kommunizieren, sondern auch mit hilfe zweier anderer Blöcke:

(Den ersten dieser beiden Blöcke haben wir bereits verwendet!)

Aufgabe: Lass dir vom **Prototypen bee** eine Liste seiner **Abkömmlinge geben!**

Die Ergebnisliste besteht aus drei Elementen - **drei Objekten:**

1) **Die Sammler-Biene**
2) **Die Drone**
3) **Die Königin**

Das sind die Abkömmlinge, d.h. **Kinder** des Prototypen.

Wir erinnern uns: Wir haben diese Objekte erzeugt, indem wir den Prototypen geklont haben. Dadurch wurden sie als Abkömmlinge erzeugt!

Indem wir nun ein zufälliges Element aus dieser Liste klonen, können wir neue Objekte erzeugen.

Aufgabe: Lass die Königin ein **zufälliges Element dieser Liste klonen!**

Lösung:


Wenn angeklickt
setze Größe auf 130 %
frage bee nach Attribut Abkömmlinge
1
2
3
Länge: 3

klone
Element beliebig von frage bee nach Attribut Abkömmlinge

Da eine Bienenkolonie nur über eine einzige Königin verfügen soll, **entfernen wir vorher jedoch die Königin aus dieser Liste.**

Aufgabe: Filtere die Königin aus der Liste!

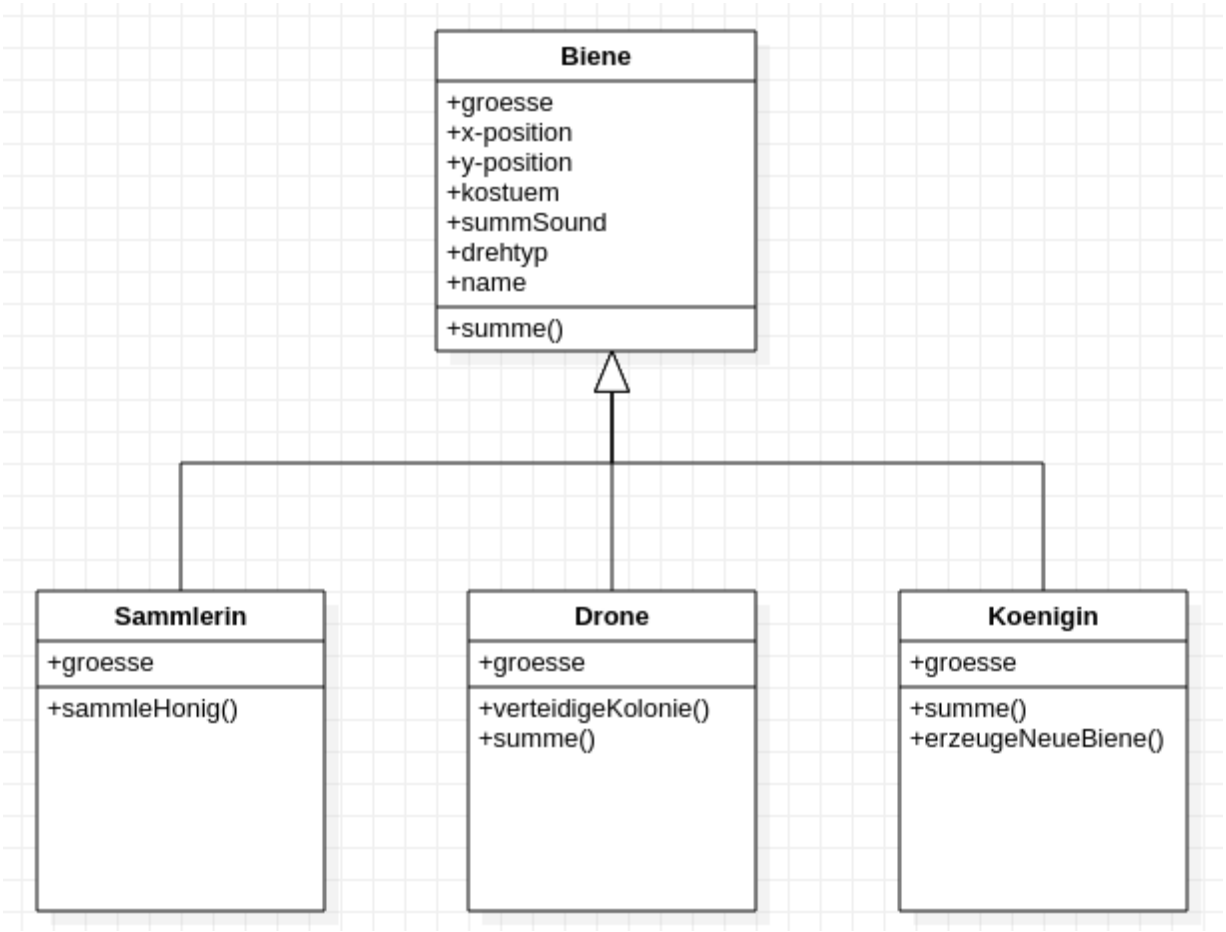
Indem wir die Abkömmlinge klonen, um von ihnen neue Kopien, bzw. Klone zu erstellen, haben wir sie effektiv zu **Prototypen** gemacht und eine weitere Abstraktionsschicht hinzugefügt.

Das bedeutet, dass wir den Code der Bienenarten nun anpassen müssen!

Aufgabe: Passe den Code der Objekte **gatherer** und **drone** folgendermaßen an:

- Bisherige Skripte sollen nicht mehr gestartet werden, wenn die grüne Flagge geklickt wird, sondern beim Ereignis "**Wenn ich als Klon starte"**
- Nach diesem neuen Ereignis-Block soll der anzeigen-Block folgen
- Beide Objekte brauchen zusätzlich jeweils ein neues Skript:
  - Wenn die grüne Flagge geklickt wird,
  - verstecke dich

Wir fassen das UML-Modell zusammen:



Lösung zum Entfernen der Königin aus der Liste:

Dieser Schritt ist komplex und muss evtl. gemeinsam mit den Schülerinnen und Schülern entwickelt werden. Nur **sehr wenige** Schülerinnen und Schüler werden diese Lösung selbstständig erreichen.

Reflektion

Was haben wir in diesem Abschnitt konzeptionell getan?

Wir haben das Verhalten einer Bienenkolonie in Snap! simuliert. Dabei haben wir die unterschiedlichen Arten von Bienen zuerst ausgehend von ihren Unterschieden zueinander definiert und diese jeweils implementiert. Alle Arten von Bienen haben unterschiedliche Eigenschaften und Verhaltensweisen des Prototypen dynamisch übernommen oder überschrieben.
Wir haben uns dabei auf Prototypen und Klone gestützt, um das Klassen-Objekt-Paradigma nachzustellen, denn Snap! kennt keine klassenbasierte Vererbung. Stattdessen haben wir eine prototypische Biene beschrieben, und Klone von ihr erzeugt, um neue Objekte zu erstellen.