

Aufgabe 1

- Erstelle ein neues Projekt in BlueJ, bestehend aus:
 - einer Klasse `Tier` mit der Methode `public void gibLaut()`, die den Text "Was bin ich?" ausgibt
 - einer Klasse `Hund` als Unterklasse der Klasse `Tier`, die die Methode `gibLaut()` überschreibt und den Text "Wau!" ausgibt
 - einer Klasse `Katze` als Unterklasse der Klasse `Tier`, die die Methode `gibLaut()` überschreibt und den Text "Miau!" ausgibt
- Erstelle mindestens ein Objekt je Klasse und teste die Methode `gibLaut()`
- Ändere bei der Klasse `Hund` den Namen der Methode von `gibLaut()` auf `giblaut()`. Kompiliere dann die Klasse mit und ohne Verwendung der Annotation `@Override`

Polymorphie und Dynamische Bindung

Aufgabe 2

- Erweitere das Projekt aus Aufgabe 1 um eine weitere Klasse `Main` mit einer Methode `public static void main()`
- Ergänze Code, der zunächst eine Referenz `Tier t` erzeugt und dann zufällig ein Objekt vom Typ `Hund` oder vom Typ `Katze` erzeugt und dieser Referenz zuweist
 - Verwende die Methode `Math.random()` zur Erzeugung von Zufallszahlen
 - Hinweis: Code der Form `Tier t = new Hund();` ist erlaubt, da jeder Hund auch ein Tier ist (Ist-Ein-Beziehung)!
- Rufe die Methode `gibLaut()` auf das Objekt `t` auf

- Wird das Programm aus Aufgabe 2 ausgeführt, so gibt es entweder “Wau!” oder “Miau!” aus, abhängig davon, ob das Objekt `t` vom Typ `Hund` oder vom Typ `Katze` ist
 - Dies entscheidet sich erst zur Laufzeit, d. h. beim kompilieren ist noch nicht bekannt, welche `gibLaut()`-Methode aufgerufen wird
 - Man bezeichnet diesen Vorgang als **Dynamische Bindung**
- Zu beachten:
 - Obwohl `t` vom Typ `Tier` ist, wird **nie** der Text “Was bin ich?” ausgegeben!

- Erst zur Laufzeit wird ermittelt, von welchem Typ ein Objekt tatsächlich ist und dann die entsprechende Methode aufgerufen
- Dieses Verhalten ist nicht in allen Programmiersprachen der Standard.

Beispiel C++:

- Bindung erfolgt beim Kompilieren anhand des angegebenen Datentyps
- Soll eine Methode dynamisch gebunden werden, muss sie explizit als **virtual** definiert werden
- Durch dynamische Bindung wird **Laufzeitpolymorphie** möglich

Polymorphie

- Griechisch für “Vielgestaltigkeit”
- Laufzeitpolymorphie/dynamische Polymorphie:
 - Hinter einer Variable vom Typ einer bestimmten Klasse können sich Objekte unterschiedlichen Typs verbergen, nämlich von sämtlichen Unterklassen dieser Klasse.
 - Methodenaufrufe hängen dann von der konkreten Klassenzugehörigkeit (zur Laufzeit) ab
 - Dies funktioniert nicht nur mit Klassen, sondern auch mit Interfaces
- Überladung/statische Polymorphie:
 - Auch das Überladen von Methoden bzw. Operatoren (gleicher Name, unterschiedliche Signatur) ist eine Form der Polymorphie
 - Hier ist bereits beim Kompilieren klar, welche Methode “gemeint” ist
- Generische Programmierung/parametrische Polymorphie
 - Verwendung generischer Klassen/Interfaces/Methoden
 - In Java: Generics (kommt noch)

- Manchmal möchte/muss man in einem Programm wissen, von welchem Typ ein Objekt tatsächlich ist
- Hierfür bietet Java den Operator `instanceof`
- Beispiel (vgl. Aufgabe 1 und 2):

```
if ( t instanceof Katze ) { ... }
```

Aufgabe 3

- Erweitere den Code aus Aufgabe 2 in `main` um eine Prüfung des tatsächlichen Typs des Objekts `t` mittels `instanceof`
 - Abhängig vom tatsächlichen Typ soll der Text “Hund” oder “Katze” ausgegeben werden