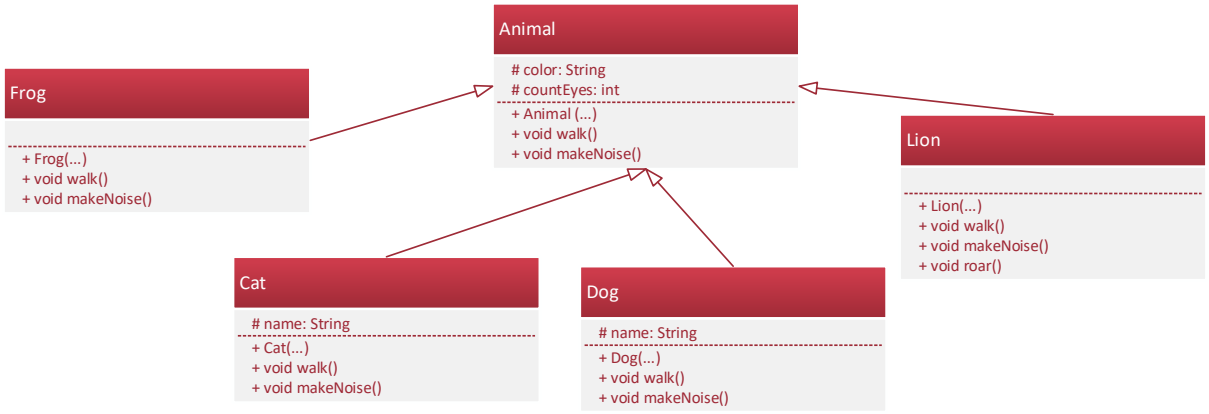
Übungsbeispiele Vererbung

1) Tiere

Bilden Sie folgende Vererbungshierarchie aus der Tierwelt ab. Erstellen Sie dazu eine Basisklasse *Animal* mit den definierten Eigenschaften und Methoden. Leiten Sie dazu die einzelnen Tiere ab und überschreiben Sie die Methoden *walk()* und *makeNoise()*, dass Sie auf der Konsole immer dem Tier entsprechende Aktionen ausgeben. Z.B. bei der Klasse Frog: *walk()* könnte „jumping“ und *makeNoise()* könnte „quak“ liefern.



Erstellen Sie in einer DemoAnwendung einzelne Objekte der Klassen und überprüfen Sie die Ausgaben.

Probieren Sie dabei folgendes Punkte aus:

- Erstellen Sie ein Objekt vom der Klasse Frog mit dem Namen quaxi
- Casten Sie quaxi zu einem Animal mit dem Namen quaxiAsAnimal
- Casten Sie quaxiAsAnimal wieder zu einem Frog
- Probieren Sie weitere casts

Erstellen Sie eine Klasse *Nature*, wo Sie in einer *ArrayList* (als Attribut) verschiedene Tiere verwalten können. Implementieren Sie eine Methode *public void addAnimal(Animal a)*, um neue Tiere in die Liste aufzunehmen. Implementieren Sie eine Methode *public int countColor(String color)*, welche die Tiere einer bestimmten Farbe zählt und diese als Ergebnis zurück liefert.

Testen Sie diese Klasse.

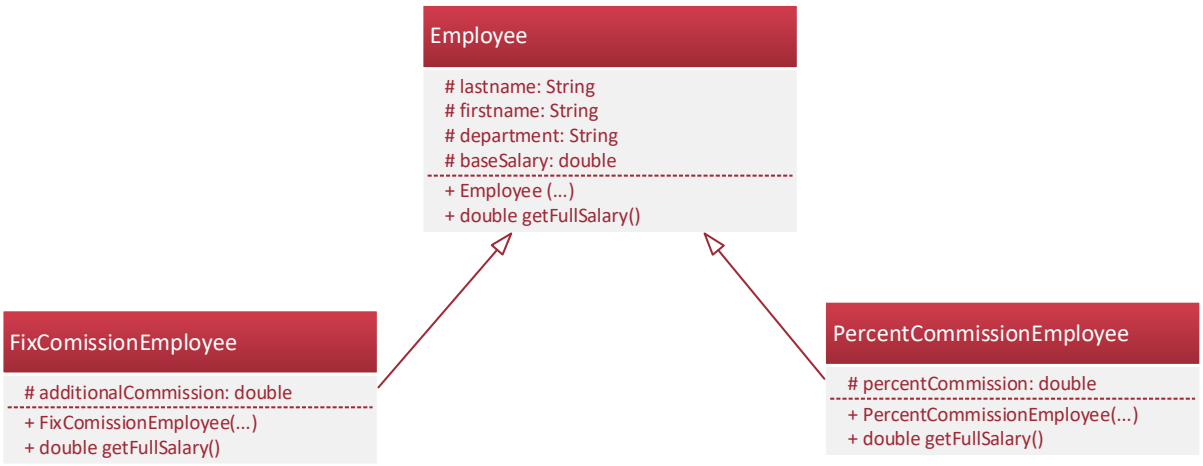
2) MitarbeitInnen

Programmieren Sie folgende Klassenhierarchie, welche verschiedene MitarbeiterInnengehälter abbildet.

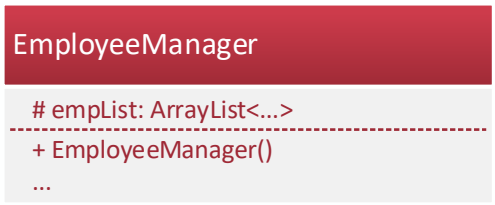
Von der Basisklasse Employee, mir ihren **protected** Attributen (siehe folgende Grafik), leiten Sie die spezialisierten Klassen FixCommissionEmployee und PercentCommissionEmployee ab.

Die spezialisierten Klassen überschreiben jeweils die Methode *getFullSalary()*, welche das monatliche Gehalt berechnet. Die Klasse *Employee* retourniert hier das Grundgehalt (baseSalary), die Klasse *FixComissionEmployee* addiert zum Grundgehalt die hinterlegte zusätzliche Provision (*additionalCommission,* zB.: Grundgehalt: 2000 Euro, Zusatzprovision: 300 Euro ➔ 2.300 Euro), die

Klasse *PercentCommissionEmployee* addiert ausgehend vom Basisgehalt die prozentuelle Provision (*percentCommission,* zB.: Grundgehalt: 2000 Euro, Prozent-Provision: 10 % ➔ 2.200 Euro). Implementieren Sie in jeder Klasse einen Konstruktor, welche die erforderlichen Attribute setzt.



Erstellen Sie eine Klasse *EmployeeManager* mit einer Liste zur Verwaltung von MitarbeiteInnen.



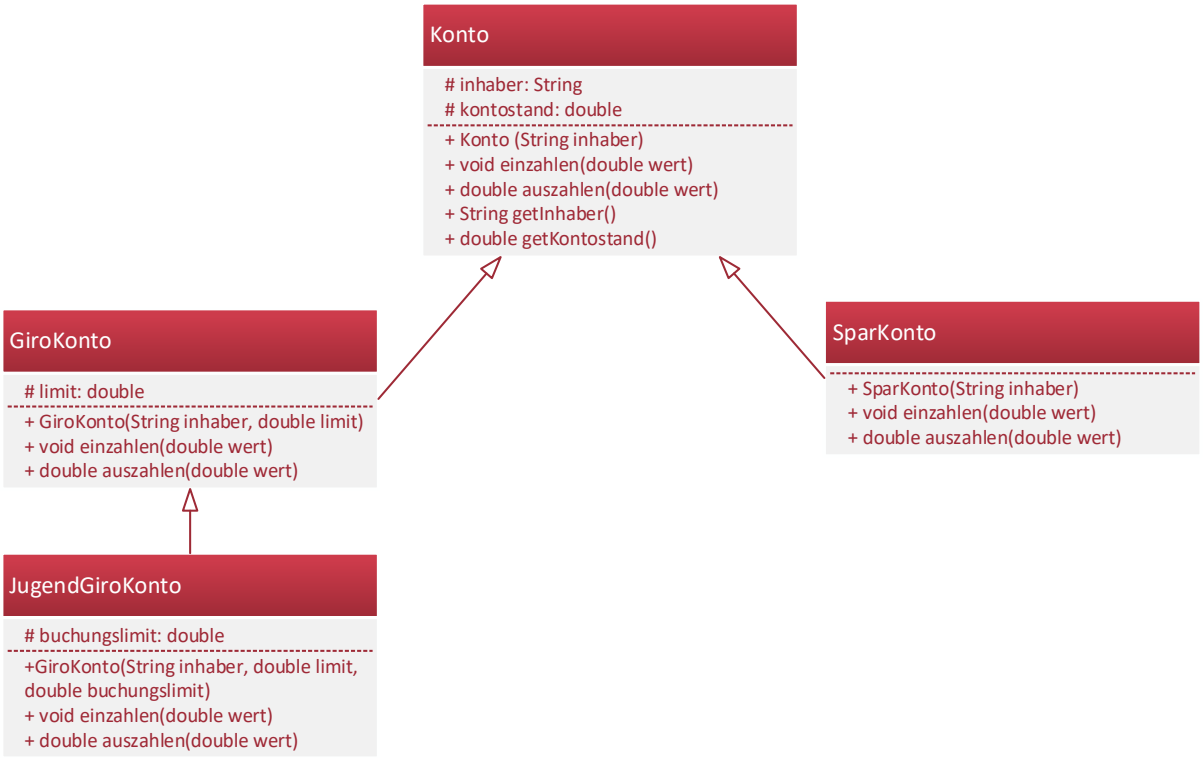
Implementieren Sie folgende Methoden

- *public double calcTotalSalary()* ➔ Diese berechnet das gesamte Gehalt aller MitarbeiterInnen (inklusive derer Provisionen).
- *public void addEmployee(…)* ➔ Diese nimmt einen neuen Employee in die Liste auf.
- *public HashMap<String, Double> getSalaryByDepartment()* ➔ Liefert das gesamte Gehalt aller Mitarbeiter auf die einzelnen Abteilungen abgebildet.

Erstellen Sie eine Testanwendung zum Testen Ihrer Klassen.

3) Konto

Bilden Sie folgende Vererbungshierarchie ab und implementieren Sie die Klassen wie folgt gegeben.



Konto: Übernimmt den Kontoinhaber im Konstruktor und setzt den Kontostand auf 0. Die Methoden *public void einzahlen(double wert)* und *public double auszahlen (double wert)* erhöhen oder verringern den Kontostand entsprechend ihrem Namen und auszahlen liefert zusätzlich die Höhe der Auszahlung als return Wert..

Girokonto: Übernimmt im Konstruktor zusätzlich ein Limit. Auszahlungen sind maximal bis zu diesem Limit möglich. Ist das Limit beispielsweise auf 1000 Euro gesetzt, so kann das Konto auf -1000 Euro überzogen werden.

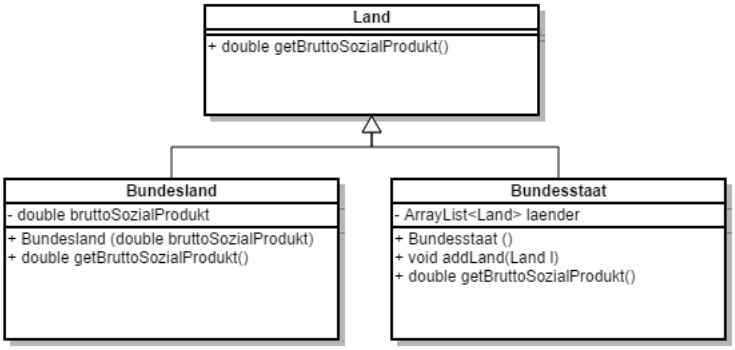
JugendGiroKonto: Übernimmt im Konstruktor zusätzlich ein Limit und ein Buchungslimit. Das Limit soll wie beim Girokonto funktionieren. Das Buchungslimit stellt sicher, dass bei einer Transaktion nur maximal das Buchungslimit abgebucht werden kann.

SparKonto: Das Sparkonto kann nicht überzogen werden und es ist nicht möglich einen negativen Kontostand zu haben.

Erstellen Sie eine Demoanwendung und testen Sie alle 4 Klassen mit ihren spezifischen Methoden.

Testen Sie folgenden Use Case:
Erstellen Sie ein Objekt vom JugendGiroKonto und casten Sie dieses in ein Konto. Rufen die Methoden *einzahlen(…)* und *auszahlen(…)* auf. Casten Se dieses Objekt nun in ein GiroKonto und testen Sie wieder beide Methoden.

4) Land



- Leiten Sie von der Klasse *Land* eine Klasse *Bundesland* ab. Diese Klasse soll auch das *BruttoSozialProdukt* des Bundeslandes über den *Konstruktor* entgegennehmen und über die zu überschreibende Methode *getBruttoSozialProdukt* retournieren.
- Leiten Sie von der Klasse *Land* eine Klasse *Bundesstaat* ab. Diese Klasse erhält kein eigenes *Bruttosozialprodukt* über den Konstruktor.
- Ein *Bundesstaat* soll Instanzen von *Land* in einer privaten *ArrayList<Land>* verwalten können.
- Spendieren Sie dem *Bundesstaat* eine Methode *add*, welche ein *Land* entgegennimmt und diese in der im letzten Punkt erwähnten *ArrayList<Land>* ablegt.
- Überschreiben Sie im *Bundesstaat* die Methode *getBruttoSozialProdukt*, sodass diese Methode die Summe der Bruttosozialprodukte sämtlicher aufgenommener Länder, welche sich in der zuvor erwähnten *ArrayList<Land>* befinden, retourniert.
- Erstellen Sie eine ausführbare Klasse, welche ein*en Bundesstaat* mit zwei Bundesländern erzeugt. Die ausführbare Klasse soll das Bruttosozialprodukt des *Bundesstaates* auf der Konsole ausgeben.