Ternärer Operator

- Der Ternäre Operator ersetzt eine if-else Verzweigung
- Setzt einen Wert in Abhängigkeit vom Ergebnis einer Bedingung
- Wird oft Bedingungsoperator genannt
- Aufbau: bedingung ? wert1 : wert2 → Wenn die Bedingung zutrifft wird wert1 zurückgeliefert, sonst wert2

Enums

- Ein Enum ist eine spezielle Klasse, die eine Sammlung von Konstanten repräsentiert
- Enums werden mit dem Schlüsselwort **enum** deklariert, zum Beispiel:

```
enum Gamelevel
{
        EASY,
        MEDIUM,
        HARD
}
```

- Der Zugriff auf das Enum erfolgt folgendermaßen:

```
Gamelevel gl = Gamelevel.HARD;
 System.out.println("All games are " + gl); // HARD
```

Exceptions

- Exceptions sind Ereignisse, die während der Ausführung eines Programms auftreten und den normalen Fluss des Programms unterbrechen.
- In Java werden Exceptions durch die Klassen java.lang.Exception und java.lang.Error repräsentiert.
- Checked Exceptions: Diese müssen explizit im Code behandelt oder weitergeleitet werden (z.B. IOException, SQLException).
- Unchecked Exceptions: Diese müssen nicht explizit behandelt werden und entstehen meistens aufgrund von Programmierfehlern (z.B. NullPointerException, ArrayIndexOutOfBoundsException).

Exceptions

- Exceptions werden mit einem try-catch-Block behandelt.
- Der try-Block enthält den Code, der eine Exception auslösen könnte.
- Der catch-Block definiert, wie auf eine bestimmte Exception reagiert werden soll.
- Ein finally-Block kann optional verwendet werden, um Code auszuführen, der nach dem try- und catch-Block ausgeführt werden soll, unabhängig davon, ob eine Exception aufgetreten ist oder nicht.
- Es gibt die Möglichkeit mit throw eine Exception zu werfen.

Gemeinsame Aufgabe

Schreibe ein Programm, dass den User nach Zahlen fragt, von denen am Ende der Durchschnitt berechnet werden soll. Solange der User während der Eingabe keine gültige Zahl eingibt, soll er die Eingabe wiederholen.

Das Programm darf bei falscher Eingabe nicht abstürzen.

Vererbung

- Vererbung bezieht sich in Java auf Klassen, die von anderen Klassen abgeleitet werden
- Wird eine Klasse von einer anderen abgeleitet, erbt die abgeleitete Klasse alle Attribute und Funktionen der „Überklasse", die NICHT als private deklariert waren
- Die Ableitung führt man herbei in dem man in der Klassendeklaration das Schlüsselwort extends verwendet und danach die Überklasse angibt.

Z.B.:

```
public class Car extends Vehicle {}
```

Vererbung

Eine Subklasse kann somit weitere Funktionen und Attribute zu der der Überklasse besitzen

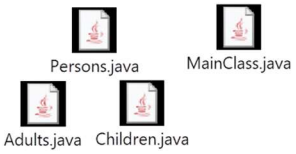
Somit bietet die Überklasse eine Art Basis für alle Unterklassen

Auch der Konstruktor der Überklasse wird vererbt

Besitzt eine Überklasse nicht den Basiskonstruktor (den leeren) muss in der Unterklasse ein eigener Konstruktor erstellt werden, in dem der Konstruktor der Überklasse aufgerufen wird. Dies passiert über den **super** Befehl

Z.B.:

```
public Car(double hp, double pr, String c)
{
        super(hp, pr, c);
}
```



Abstrakte Klassen

- Abstrakte Klassen sind „normalen" Klassen sehr ähnlich.
- Sie können Funktionen und Attribute enthalten.
- Diese werden mit dem Keyword **abstract** deklariert
- Abstrakte Klassen können NICHT instantziert werden. Man kann also mittels „new" kein Objekt daraus erzeugen
- Abstrakte Klassen dienen als „Vorgabe" für Subklassen

Interfaces

- Interfaces geben vor, wie Klassen, die dieses Interface implementieren aussehen müssen
- Dabei sind Interfaces selbst eine Klasse (vereinfachte Form einer abstrakten Klasse).
- Alle Klassen, die ein bestimmtes Interface implementieren, müssen die Funktionen implementieren, die dieses Interface beinhaltet.
- Alle Member eines Interfaces müssen, public und können static und können final sein. Andere Identifier sind nicht erlaubt.
- Innerhalb eines Interfaces können Funktionen eine default Implementierung haben. Klassen, die diese Funktion nicht implementieren besitzen automatisch die default Implementierung dieser Funktion -> ABER DAS IST BAD PRACTICE.

Interfaces

- Die Syntax eines Interfaces sieht wie folgt aus:
- `public interface Animal {}`
- Die Syntax der Implementierung eines Interfaces sieht wie folgt aus:
- `public class Dog implements Animal{}`

Instance Of

In Java ist es möglich, dass man ein Objekt mit einer Klasse deklariert, aber mit einer anderen instanziert. Z.B.:

Vehicle myVehicle = new Car(hp, price, color);

Das macht in sofern Sinn, dass man z.B. in einem Array alle Variablen vom Typ "Vehicle" speichern möchte.

Um unterscheiden zu können, mit welchem Typ diese Variablen dann instanziert wurden, wird der Befehl

Z.B.: `if(myVehicle instanceof Car) ...`

Static (Early) Binding vs. Late Binding

Early Binding bedeutet in Java, dass zur Compilezeit feststeht von welchem Typ ein Objekt ist

Late Binding bedeutet, dass der Typ eines Objektes zur Laufzeit festgesetzt wird.

Beispiel Early Binding: `Persons pers = new Persons();`

Beispiel Late Binding: `Persons pers = new Adults();`

Cast

In Java gibt es die Möglichkeit Objekte „genauer zu spezifizieren"

Wenn man ein „Vehicle" Objekt hätte und wüsste, dass es sich dabei um ein „Car" Objekt handelt könnte man das „Vehicle" Objekt in ein „Car" Object casten.

Wenn es in „Car" eine Funktion gäbe, könnten wir diese ohne Cast nicht aufrufen, da wir laut Deklaration ein „Vehicle" haben. Nach einem Cast könnten wir die Funktion verwenden.

Z.B.: `((Car)myVehicle).startTheEngine();`

Pattern Matching

- Der Traditionelle Weg den Datentyp einer Subklasse zu ermitteln wäre folgender:

```
if(pers instanceof Children)
{
  System.out.println("Here is a child: "+ ((Children)pers).getSchool());
}
else if(pers instanceof Adults)
{
   ((Adults)pers).getAge();
   System.out.println("Here is an adult");
}
```

Pattern Matching

- Diese traditionelle Methode benötigt einen cast, um die Funktionen der spezifischen Klasse aufzurufen.
- Mittels Pattern Matching kann man auf den cast verzichten:

```
if(pers instanceof Children child)
 {
   System.out.println("Here is a child: "+ child.getSchool());
 }
```

Sealed-permits, non-sealed, final Klassen und Interfaces

- Manchmal möchte man bestimmten Klassen eine Ableitung ermöglichen, manchmal nicht.
- Zum Beispiel könnte man eine Klasse „Vehicles" errichten, der man gewisse Attribute zuordnet, die für Autos und LKWs passen, aber nicht für Boote
- Wird eine Klasse mit dem Keyword sealed deklariert, muss darauf auch das Keyword permits folgen.
- Sealed verbietet das Ableiten einer Klasse, permits definiert die Ausnahmen
- Wenn diese Ausnahmen definiert werden müssen sie als Identifier entweder die Keywords final oder non-sealed beinhalten.
- Das Keyword final in einer Klasse bedeutet, dass diese Klasse nicht mehr abgeleitet werden darf, non-sealed bedeutet, dass die Klasse weiterhin abgeleitet werden darf

Sealed-permits, non-sealed, final Klassen und Interfaces

```
public sealed interface Animal permits Dog,
Cat
    public final class Dog implements Animal
    public non-sealed class Cat implements
    Animal{ // maybe derive tiger, gepard, ..
```