Datenstrukturen: Allgemeine Konzepte und die Klasse `node`

Wir kennen und benutzen bereits einige Datenstrukturen, z.B. Graphen und Arrays. Als Informatiker sollten wir erstens einen soliden Grundstock an Strukturen kennen (und verstehen) und zweitens sollten wir Möglichkeiten kennen, wie wir solche Strukturen verallgemeinern können.

Es geht nun zunächst um Verallgemeinerungen. Warum überhaupt verallgemeinern? Weil es ineffizient ist, eine Graph-Struktur einmal für Raum-Objekte eines Adventures und ein anderes Mal für Spielfeld-Objekte eines Brettspiels zu programmieren. Sinnvoller ist es, wenn eine Graph-Struktur sowohl für Raum-Objekte als auch für Spielfeld-Objekte identisch funktioniert. Wir betrachten eine Graph-Struktur, bei der jeder Knoten vier Kanten kennt.

Eine Strategie ist es, eine allgemeine node-Hilfsklasse zu benutzen:

```
public class node {
    node north, west, east, south;
    Object inhalt;
    boolean visited;
    // ..... (Methoden fehlen ggf. noch)
}
```

Wichtig ist die Eigenschaft **`inhalt`** vom Typ **`Object`**. Dazu müssen wir uns wieder an das Konzept „Vererbung“ erinnern: Mit dem **`extends`**-Schlüsselwort kann man anzeigen, dass z.B. eine Klasse **`Tiger`** von einer Klasse **`Tier`** „erbt“, also eine Unterklasse von **`Tier`** ist. In Java gibt es eine Besonderheit: Jede Klasse ist immer Unterklasse der Klasse **`Object`**. Dadurch kann die Eigenschaft **`inhalt`** beliebige Objekte aufnehmen! Diese Vielseitigkeit bringt aber auch Probleme mit sich. Betrachte folgende Zeilen:

```
Object irgendwas = new Point(1,3);   // klappt
Point p = irgendwas;    // Fehlermeldung!
```

Das Objekt **`irgendwas`** ist zunächst vom Typ **`Object`**. Aber weil nicht jedes **`Object`**-Objekt vom Typ **`Point`** ist, funktioniert die zweite Zeile nicht. Weil wir (als Programmierer) aber in diesem Fall wissen, dass hinter **`irgendwas`** tatsächlich ein **`Point`**-Objekt steckt, kann man explizit schreiben:

```
Point p = (Point) irgendwas;    // klappt!
```

Das klappt aber natürlich nur, falls **`irgendwas`** wirklich mal ein **`Point`** war! (Dies kann man abfragen mit **`(irgendwas instanceof Point)`**). Diese explizite Typ-Umwandlung nennt man **„Type-Cast“**.

Aufgabe: Schreibe ein Testprogramm, welches mit Hilfe der **`node`**-Klasse zwei verschiedene Graphen erstellt (z.B. einen Graphen mit **`String`**-Objekten als Inhalte und einen anderen mit **`Point`**-Objekten als Inhalte). Schreibe außerdem eine rekursive Methode **`besuche(node k)`**, die den Graphen durchläuft und dabei jedes Objekt einfach mit **`System.out.println`** ausgibt. **`besuche`** soll für beide Graphen-Typen funktionieren!