Wo sind wir?

- Java-Umgebung
- Lexikale Konventionen
- Datentypen
- Kontrollstrukturen
- Ausdrücke
- Klassen, Pakete, Schnittstellen
- **JVM**
- Exceptions
- Java Klassenbibliotheken
- Ein-/Ausgabe
- Collections
- Threads
- Applets, Sicherheit
- Grafik
- Beans
- Integrierte Entwicklungsumgebungen

JVM im Detail

Die JVM wird aufgerufen mit einem Klassennamen:

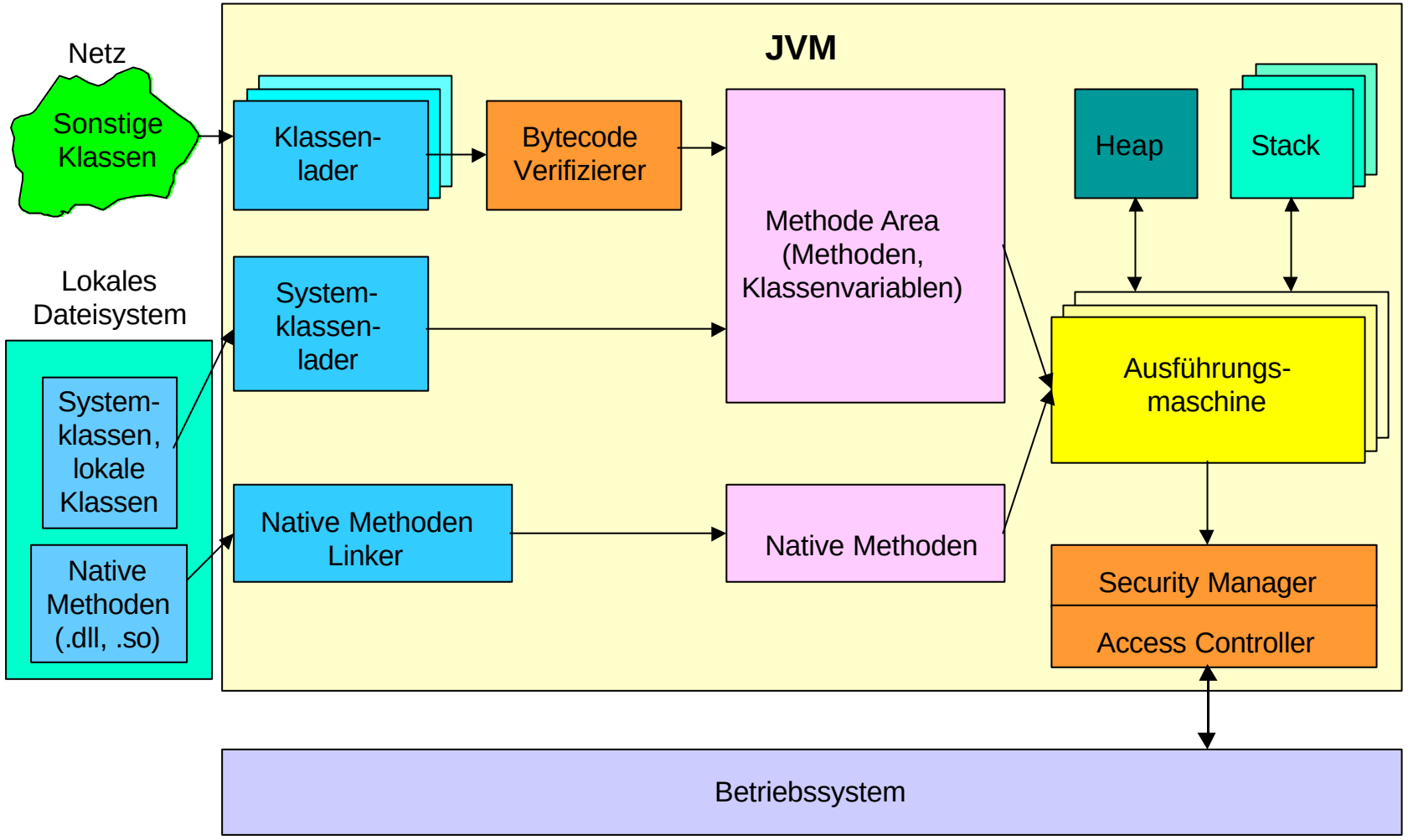
```
java classname
```

In dieser Klasse wird nach einer Methode `main()` (bzw. bei Applets nach `init()`) gesucht.

Kommt die JVM bei der Abarbeitung des Codes auf eine Klasse, die noch nicht geladen ist (so auch zu Beginn die Startklasse), so werden folgende Schritte ausgeführt:

- Über den Klassenlader wird die Binärdarstellung (.class; auch andere Formate möglich) der Klasse mit dem vorgegebenen Namen (vollqualifizierter Name) gesucht und geladen.
- Nach dem Laden erfolgt das Binden der Klasse.
- Anschließend erfolgt die Initialisierung der Klasse.

Interner Aufbau der JVM



Laden von Klassen durch den Klassenlader

- Handelt es sich um eine Klasse des lokalen Dateisystems, so wird diese Klasse dort gesucht (unter Einbeziehung von CLASSPATH und Paketstruktur). Bestehen die notwendigen Zugriffsrechte zum Lesen der Datei (z.B. unter Unix Leserechte des Verzeichnisses und der Datei), so wird die Datei durch den Klassenlader gelesen und geladen. Die Überwachung ist nicht Teil der JVM, sondern des darunter liegenden Betriebssystems.

- Muss die Datei über eine Netzwerkverbindung geladen werden, so handelt es sich prinzipiell um eine "untrusted" Klasse, die angefordert wird und anschließend die Sicherheitsüberprüfungen der JVM durchlaufen muss, bevor sie genutzt werden kann.

Binden

Nach dem Laden folgt das Binden der Binärdarstellung.

- Zuerst wird verifiziert, dass die vorliegende Binärdarstellung eine korrekte Klasse ist (Sprungziele nur innerhalb der Klasse, gültige Operationscodes usw.)

- Anschließend wird die Auflösung von symbolischen Referenzen vorbereitet. Dazu gehört das Anlegen und Initialisieren von Klassenvariablen mit Default-Werten.

- Zuletzt werden die symbolischen Referenzen aufgelöst. Dies bedeutet, dass Zugriffe auf symbolische Referenzen (vollqualifizierte Namen) in die entsprechenden Zugriffe auf die Datenstrukturen der Klassen im Speicher ersetzt werden.
  Beispiel:
  Der Zugriff auf `Integer.MAX_VALUE` wird durch einen Zugriffsoperation auf die entsprechende Speicherstelle ersetzt, wo die Klassenvariable `Integer.MAX_VALUE` zu finden ist.

Initialisierung einer Klasse

Nachdem eine Klasse gebunden ist, kann sie initialisiert werden. Dazu werden alle Initialisierer von Klassenvariablen und statische Blöcke (klassenbezogene Initialisierer) in ihrer textuellen Reihenfolge im Programm ausgeführt.

Bevor die Initialisierer ausgeführt werden können, muss jedoch (falls vorhanden) die direkte Oberklasse (rekursiv) initialisiert werden. Dies kann dazu führen, dass die Oberklasse selbst wieder geladen, gebunden und initialisiert werden muss, dies kann weitere Ladevorgänge auslösen usw.

Erzeugen einer Klasseninstanz

Wird eine neue Klasseninstanz erzeugt, z.B. durch

```
MeineKlasse o = new MeineKlasse();
```

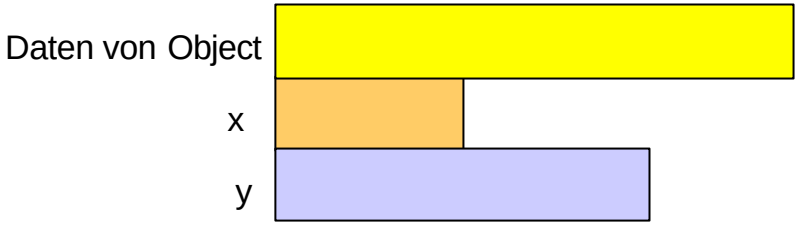
So werden folgende Schritte ausgeführt:

- Anlegen von Speicherplatz auf dem Heap durch die JVM. Der Speicherplatz nimmt alle Instanzvariablen der Klasse und aller Oberklassen auf.
- Initialisieren der Instanzvariablen (einschließlich der Oberklassen) mit ihren Default-Werten.
- Aufruf des Konstruktors mit der vorgegebenen Signatur. Im Konstruktor werden eventuell die Werte von Instanzvariablen überschrieben (explizite Initialisierungsausdrücke).

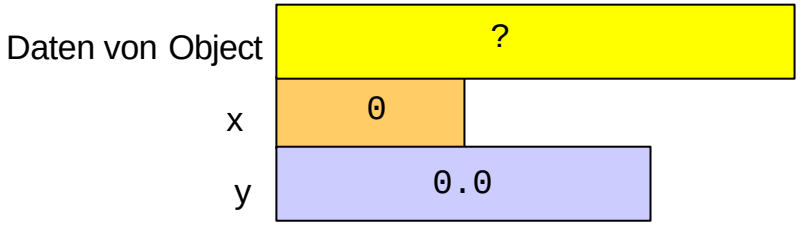
Beispiel

```
class K {
    int x = 3;
    double y;

    K() {
      System.out.println("neu");
    }
}

class Test {
    K o = new K();
}
```

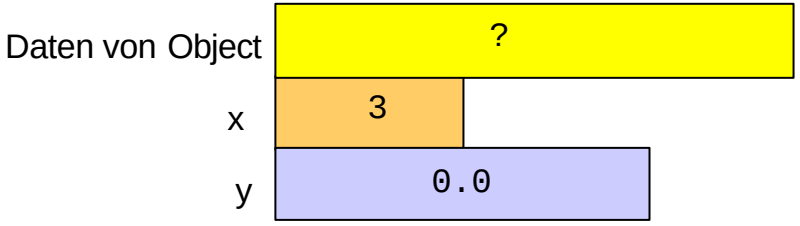
1) Anlegen von Speicherplatz



2) Initialisieren mit Default-Werten



3) Konstruktor



Garbage Collector (1)

Objekte (außer einfachen Typen) werden auf dem Heap neu erzeugt. Der Heap hat nur eine beschränkte Größe. Werden die Objekte irgendwann auch einmal gelöscht? Und wenn ja, wer löscht die Objekte und welche Objekte?

Antwort: Garbage Collector (Müllsammler)

Wenn die JVM erkennt, dass nur noch wenig Speicher auf dem Heap verfügbar ist, wird der Garbage Collector (GC) aufgerufen. Mann kann das auch aus einem Programm heraus jederzeit explizit erzwingen:

```
System.gc();                // ruft den Garbage Collector auf
```

Garbage Collector (2)

Der Garbage Collector sucht nach Objekten, auf die es keine Referenzen mehr gibt. Diese können problemlos gelöscht werden, d.h. der Speicher wieder als frei markiert werden. Es gibt verschiedene Verfahren um zu erkennen, welche Objekte nicht mehr referenziert werden (z.B. über einen Reference Count).

Beim Löschen von Objekten kann es zur Zerstückelung des Speichers kommen. Der Garbage Collector versucht deshalb gleichzeitig mit dem Löschen, den Speicher zu kompaktifizieren.

Bevor der Garbage Collector ein Objekt aus dem Speicher löscht, wird die Methode `finalize()` des Objekts aufgerufen. In der Klasse `Object` ist diese Methode definiert (die dort nichts tut) und wird somit von jedem Objekt geerbt. Man kann sie explizit in einer Klasse durch eine Methode `finalize()` überschreiben.

Beispiel

```
class MeineKlasse {
    String meinName;

    MeineKlasse(String s) {
        meinName = s;
        System.out.println("Ich bin erzeugt worden: " + s);
    }

    protected void finalize() {
        System.out.println("Ich werde gelöscht: " + meinName);
    }
}
```

```
class Test {
    static void meinTest() {
        // Objekt erzeugen
        MeineKlasse o = new MeineKlasse("Objekt 1");
        // Durch Verlassen dieser Methode gibt es keine Referenz mehr auf o
    }

    public static void main(String[] args) {
        meinTest();                 // Objekt erzeugen
        System.gc();                // Garbage Collector explizit aufrufen
    }
}
```