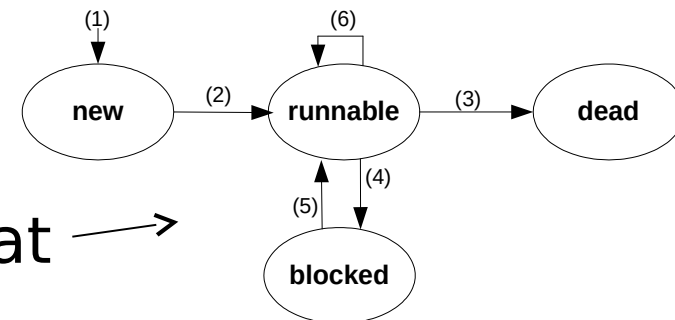


Threads in Java, JVM und Threads

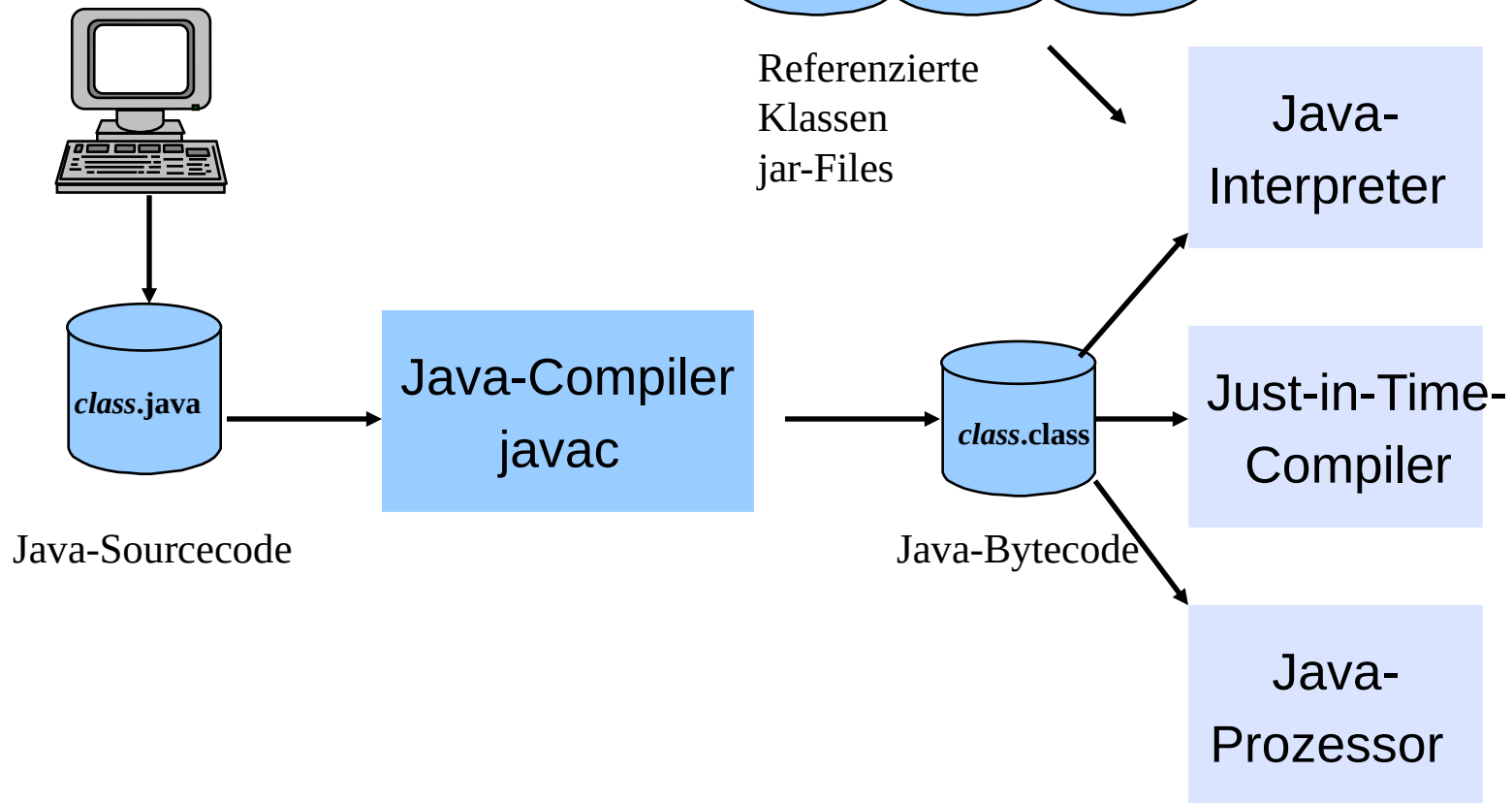
- Für jedes Programm wird eine eigene JVM gestartet
- JVM läuft in einem Betriebssystemprozess
 - Siehe z.B. im Windows Task Manager
- JVM unterstützt Threads
- Package **java.lang**
- Basisklasse **Thread**
- Vereinfachter Zustandsautomat →



- (1) Konstruktoraufwurf der Klasse Thread
- (2) Aufruf der Methode run()
- (3) Aufruf der Methode stop()
- (4) Aufruf der Methode sleep()
- (5) Aufruf der Methode resume()
- (6) Aufruf der Methode yield()

Einschub: Übersetzungsvorgang und Ablauf eines Java-Programms

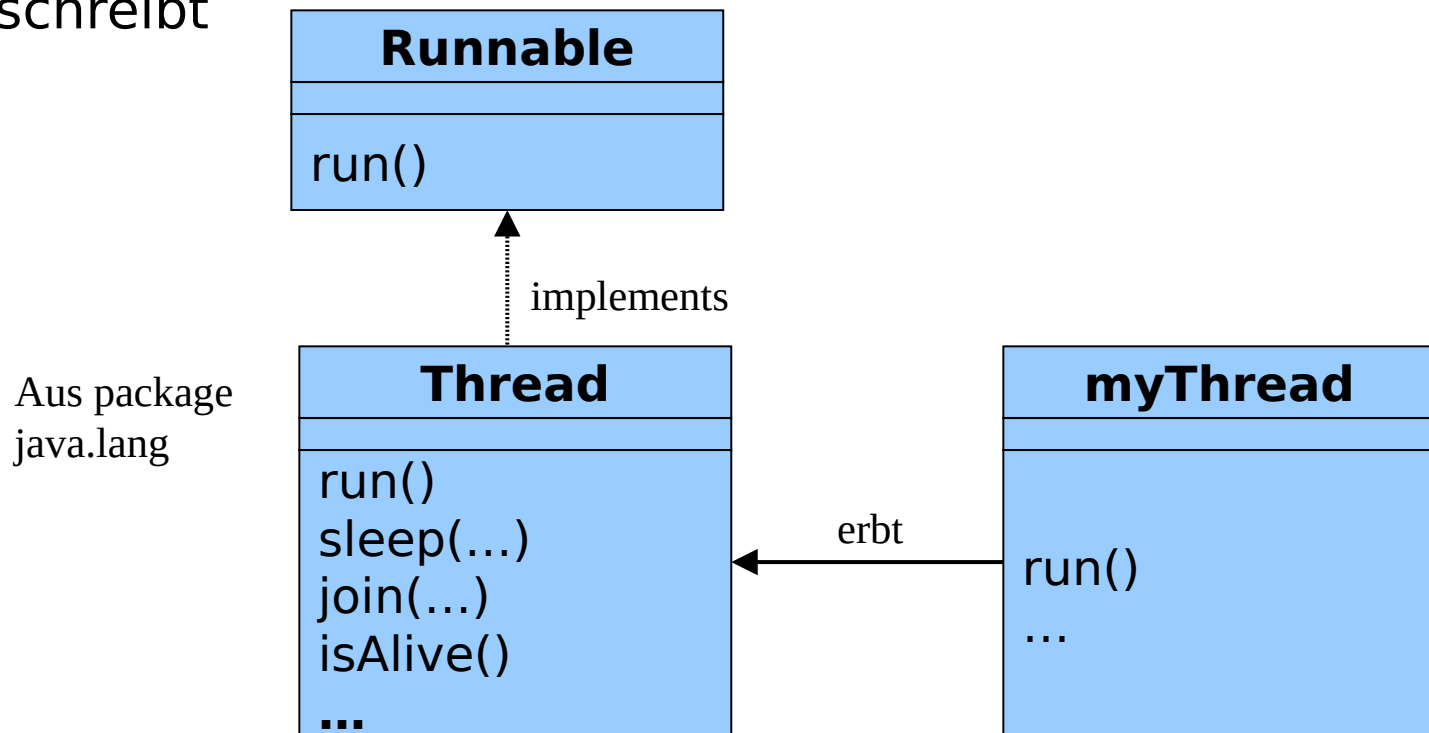
Entwicklungsumgebung:
Eclipse, ...



Threads in Java

Die Klasse Thread und das Interface Runnable

- Nebenläufigkeit wird durch die Klasse *Thread* aus Package `java.lang` unterstützt
- Eigene Klasse definieren, die von *Thread* abgeleitet ist und die Methode `run()` aus Interface *Runnable* überschreibt



Threads in Java

Beispiel: Eine einfache Thread-Klasse ...

```
import java.lang.Thread;

class myThread extends Thread { // meine Thread-Klasse

    String messageText;

    public myThread(String messageText)
    {
        this.messageText = messageText;
    }

    public void run()           // Methode, welche die eigentliche Aktion
                                // ausführt, definiert in Interface Runnable
    {
        for (;;) {
            System.out.println("Thread " + getName() + ": " + messageText);
            try {
                sleep(2000);
            }
            catch (Exception e) { /* Exception behandeln */ }
        }
    }
}
```

Threads in Java

Beispiel: ... und deren Nutzung

```
public class myThreadTest {  
  
    public static void main(String args[])  
    {  
        myThread t1;  
        t1 = new myThread("...auf und nieder immer wieder...");  
        t1.start();  
        if (t1.isAlive()) {  
            for (int i=0; i < 100000000; i++) {} // nonsense  
            try {  
                t1.join(10000);  
            } catch (InterruptedException e) { /* Exception behandeln */ }  
  
            System.out.println("Mainprogramm stoppt Thread myThread!!!");  
            t1.stop(); // deprecated  
            System.out.println("Thread " + t1.getName() + " beendet");  
        }  
    }  
}
```

- Was passiert in diesem Programm?

Threads in Java

Beispiel: Erläuterungen

- Innerhalb der Methode *start()* wird automatisch die *run()*-Methode des *Runnable*-Objekts aufgerufen
- Die Methode *join()* ohne Parameter wartet bis der Thread „stirbt“, *join(long millis)* wartet „millis“ Millisekunden und dann wird weiter gemacht
- Weitere Methoden der Klasse Thread:
 - *getPriority()*: Thread-Priorität ermitteln
 - *isAlive()*: Prüfen, ob Thread lebt
 - *getThreadGroup()*: Thread-Gruppe des Threads ermitteln
 - *interrupt()*: Thread unterbrechen
 - *getName()*: Thread-Namen ermitteln
 - ...
 - Mehrere Konstruktoren

Einschub: System-Threads

- Threads sind in Java als Gruppen hierarchisch organisiert:
 - Thread-Gruppe **system** für die Threads des Systems (der JVM)
 - Thread-Gruppe **main** für die benutzerspezifischen Threads als Untergruppe von system
- Threads der Gruppe **system**:
 - **Finalizer**: Ruft für freizugebende Objekte die finalizer-Methode auf
 - ...
 - Signal dispatcher

Einschub: System-Threads

- Weitere Threads:
 - **Garbage Collection**: hat sehr niedrige Priorität (niedriger als Idle-Thread, wartet auf Signal von Idle-Thread)
 - **Idle**: Wenn er läuft, setzt er ein Kennzeichen, das der **Garbage Collection** Thread als Startsignal betrachtet, um etwas zu tun
 - **Idle** wird nur aufgerufen, wenn die JVM sonst nichts zu tun hat