Applets

- Applets sind "kleinere" Java-Programme, die direkt in (gängigen) Internet-Browsern oder mit dem Applet-Viewer ausgeführt werden;
- benötigen keine `main`-Methode;
- müssen Unterklasse von `Applet` aus dem Paket `java.applet` sein;
- werden mittels eines `<APPLET>`-Tags in HTML-Seiten eingebunden oder mit dem Kommando `appletviewer` *`datei.html`* betrachtet.

Applets und HTML-Dokumente

```
<HTML>
<HEAD>
<TITLE> Ein Applet-Tester </TITLE>
</HEAD>
<BODY>
Hier ist das Applet:
<br>
<APPLET code = "HalloInternet.class"
        width = "200" height = "200">
</APPLET>
<br>
Ende des Applets ...
</BODY>
</HTML>
```

Wichtige Methoden der Applet-Klasse

`init()`	automatisch ausgeführt nach dem Laden des Applets
`start()`	nach `init`, Reload oder Deiconfizierung des Viewers
`stop()`	beim Verlassen der Internet-Seite (auch bei einem Reload vor `start()`) oder Iconfizierung des Viewers
`destroy()`	beim Entfernen des Applets aus dem Viewer
`paint(Graphics g)`	zeichnet das Applet-Fenster im Graphik-Kontext `g` neu

```
public void paint (Graphics g) {
    g.drawString("Hallo Internet", 50, 50);
}
```

Achtung: `Graphics` ist eine Klasse im Paket `java.awt`.

Einlesen von Parametern

Die Interaktion erfolgt mit dem HTML-Dokument, nicht mit dem Nutzer. Dadurch kann aber der Effekt des Applets angepasst werden, ohne die `.class`-Datei zu verändern.

```
<APPLET CODE = "..." WIDTH = "..." HEIGHT = "...">
<PARAM NAME = "..." VALUE = "...">
</APPLET>
```

Das Einlesen erfolgt mit Hilfe der Methode `String getParameter(String)`.

Nebenläufigkeit (Threads) – Prinzip

wie Programme: ein Thread ist eine sequentielle Folge von Anweisungen

wie Prozesse: mehrere Threads des gleichen Programms können parallel zueinander laufen

anders als Prozesse:

- Threads des gleichen Programms benutzen gleichen Speicherbereich (das kann zum Informations-Austausch genutzt werden)
- Zusammenspiel der Threads ist im Programm zu organisieren

Threads in Java

- Threads sind Objekte der Klasse `java.lang.Thread`
- jedes Java-Programm ist implizit ein Thread

  Daher kann auf alle statischen Methoden der Klasse `java.lang.Thread` zugegriffen werden.

  Beispiel:

  ```
  Thread.sleep(500);      //  in beliebigen Programmen
  ```

- Programm läuft bis alle Threads beendet sind

Wichtige Methoden

`void run()`	"Hauptprogramm" des Thread; enthält den eigentlichen Anweisungsteil, der nach Aktivierung des Thread ausgeführt wird.
`void start()`	startet den Thread mit Aufruf von `run()` (`run()` niemals direkt aufrufen!)
`Thread Thread.currentThread()`	liefert aktuellen Thread
void Thread.sleep(long ms)	läßt aktuellen Thread `ms` Millisekunden pausieren
`void join(Thread t)`	wartet auf Ende von Thead `t`
`void yield()`	läßt anderen Threads den Vortritt

Erzeugen von Threads

Zwei Wege um Threads zu erzeugen:

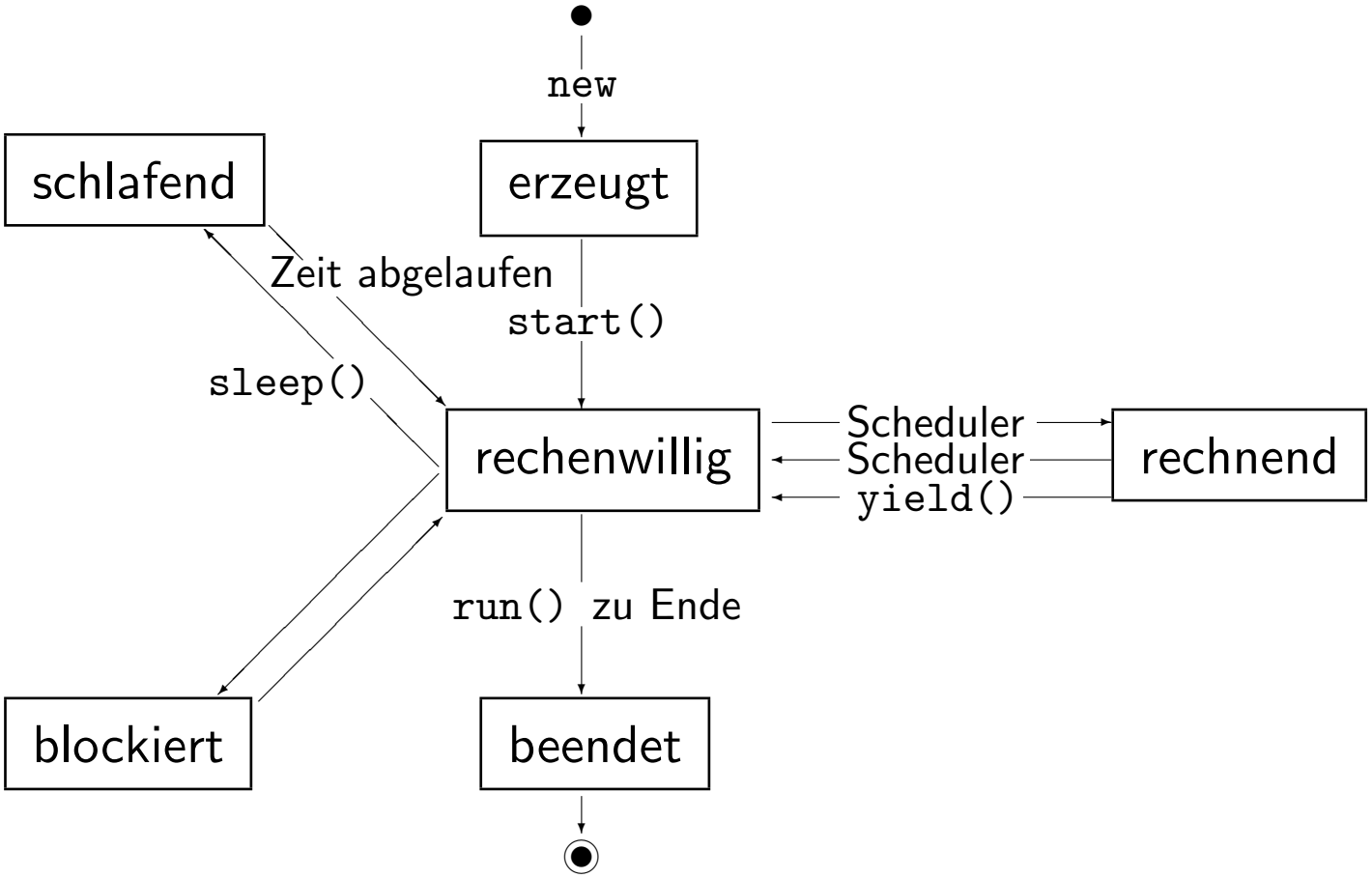
1) Unterklasse von `Thread`

```
class T
  extends Thread {
  ...
  public void run() { ... }
  ...
}
class XYZ {
  ...
  T t = new T(...);
  t.start();
}
```

2) Implementation von `Runnable`

```
class R extends X
  implements Runnable {
  ...
  public void run() { ... }
  ...
}
class XYZ
  ...
  Thread t =
    new Thread(new R(...));
  t.start();
}
```

Zustände von Threads



Synchronisation – Monitore

Schlüsselwort `synchronized` reserviert ein Objekt `obj` für einen Thread bis der Anweisungsblock (geschützter Bereich) verlassen wird:

```
...
synchronized (obj) { ... }
...
```

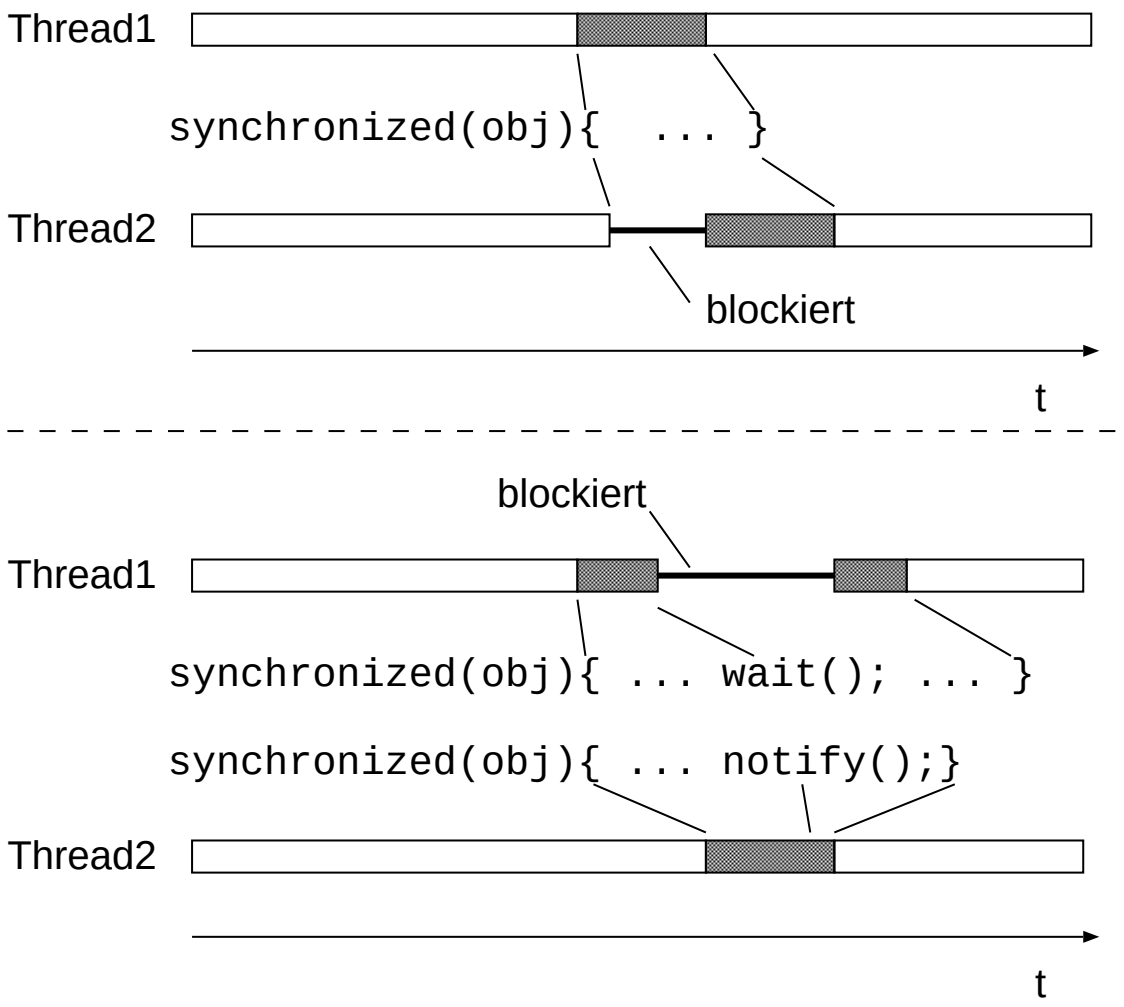
Oder:

```
synchronized ResultType method(...) { ... }
```

ist Kurzform für

```
ResultType method(...) {
  synchronized (this) { ... }
}
```

Andere Threads, die `obj` reservieren wollen, werden blockiert.


Thread1
synchronized(obj){ ... }
Thread2
blockiert
t
blockiert
Thread1
synchronized(obj){ ... wait(); ... }
synchronized(obj){ ... notify();}
Thread2
t

Synchronisation – Datenübergabe

Wenn `obj` reserviert ist, dann

`obj.wait()`	löst Reservierung und blockiert aktuellen Thread
`obj.notify()`	löst Blockierung des wartenden Threads

Beispiel:

```
synchronized (obj) {
  ...              // obj vorbereiten
  obj.wait();      // auf Ergebnis in obj warten
  ...              // Ergebnis weiterverarbeiten
}
...
synchronized (obj) {
  ...              // rechnen, Ergebnis in obj ablegen
  obj.notify();    // Fertigmeldung
}
```