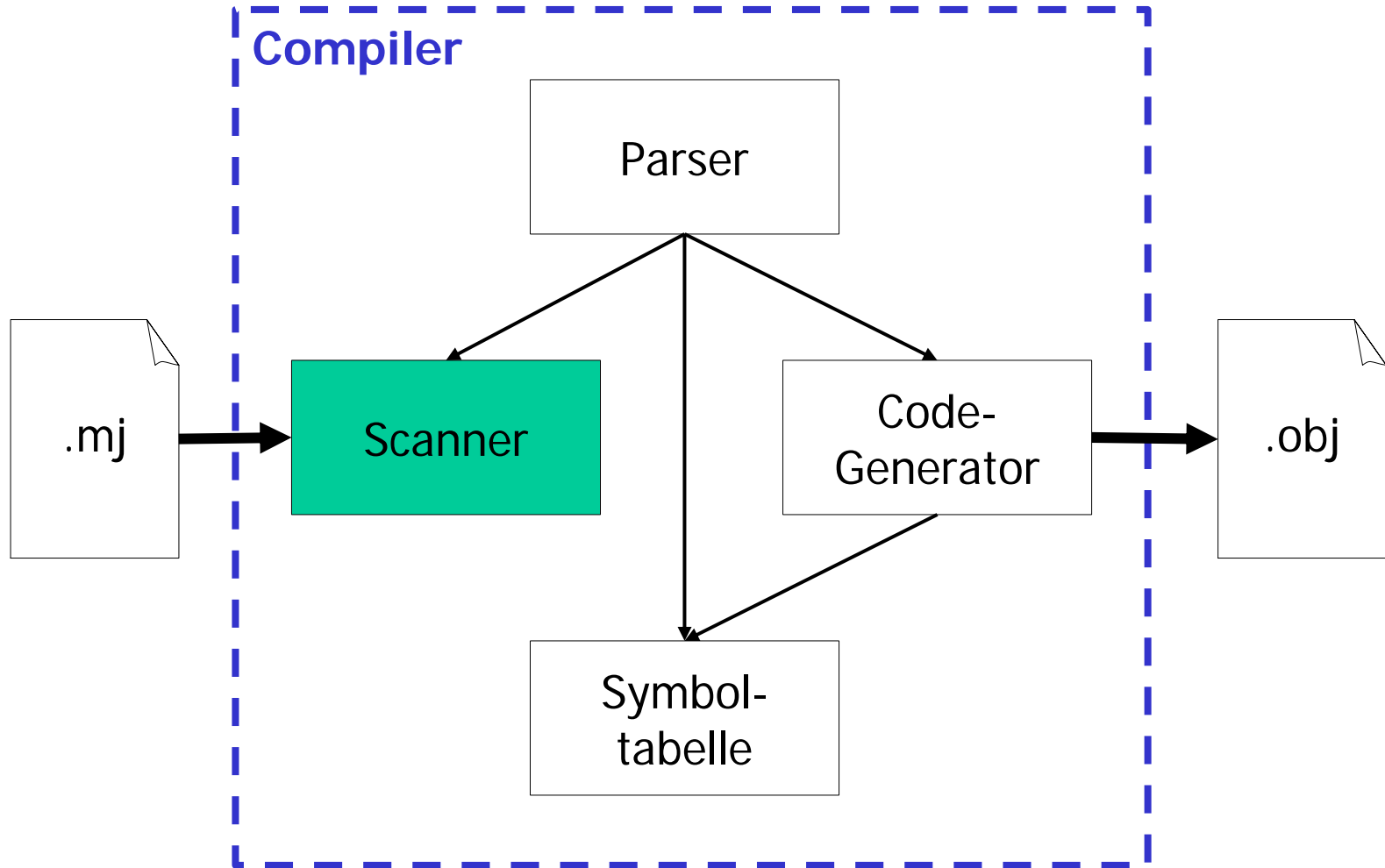


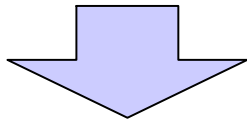
# Struktur des *MicroJava*-Compilers



# Grammatik ohne Scanner

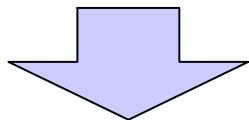


Expr = Term { "+" Term } .  
Term = Factor { "\*" Factor } .  
Factor = ident .



erlaube Kommentare an beliebiger Stelle

Expr = {Comment} Term {Comment}  
          { "+" {Comment} Term {Comment} } .  
Term = {Comment} Factor {Comment}  
          { "\*" {Comment} Factor {Comment} } .  
Factor = {Comment} ident {Comment} .



erlaube Whitespace an beliebiger Stelle

...

# Klasse Token



```
class Token {  
    int      kind;    // z.B. ident, assign, ...  
    int      line;    // Zeilenposition  
    int      col;     // Spaltenposition  
    int      val;     // numerischer Wert für number und charConst  
    String   str;     // Name von ident, string von number  
}
```

# Aufgaben des Scanners



- Erkennen von Terminalsymbolen
- Überlesen unbedeutender Zeichen (Blanks, Tabs, Zeilenumbrüche, ...)
- Überlesen von Kommentaren
- Erkennen von:
  - Namen
  - Schlüsselwörtern
  - Zahlen
  - Zeichenkonsanten
  - Stringkonstanten (gibt es in *MicroJava* nicht)
- Bildung von Terminalklassen (ident, number, ...)  
int, char, null, chr, ord, len sind **keine** Schlüsselwörter!  
nur vordeklarierte Namen (→ Erkennung als Name)
- Erkennen des Dateiendes
- lexikalische Fehler melden (Zahlenformat, ungültige Zeichen, ...)
- Einstellen der Scanner-Attribute der Tokens (Symbolart, Position, Wert, ...)

# Scanner.next() (1)



```
public static Token next () {
    while (Character.isWhitespace(ch)) nextCh();           // skip white space

    Token t = new Token();
    t.line = line; t.col = col;                            // set position

    switch (ch) {
        //----- ident or keyword
        case 'a': case 'b': ... case 'z':
        case 'A': case 'B': ... case 'Z':
            readName(t);    // distinguish between identifier and keyword
            break;
        //----- number
        case '0': ... case '9':
            readNumber(t);
            break;
        . . .
    }
```

# Scanner.next() (2)



```
...
//----- character constant
case '\':
    t.kind = Token.charConst;
    nextCh();
    if (ch == '\') { error ("empty character constant"); nextCh(); }
    else {
        if (ch >= ' ' && ch <= '~')                // printable chars
            t.val = (int) ch;
        else error("invalid character constant '" + ch + "'");
        nextCh();
        if (ch == '\') nextCh();
        else error ("missing ' at end of character constant");
    }
    break;
...
```

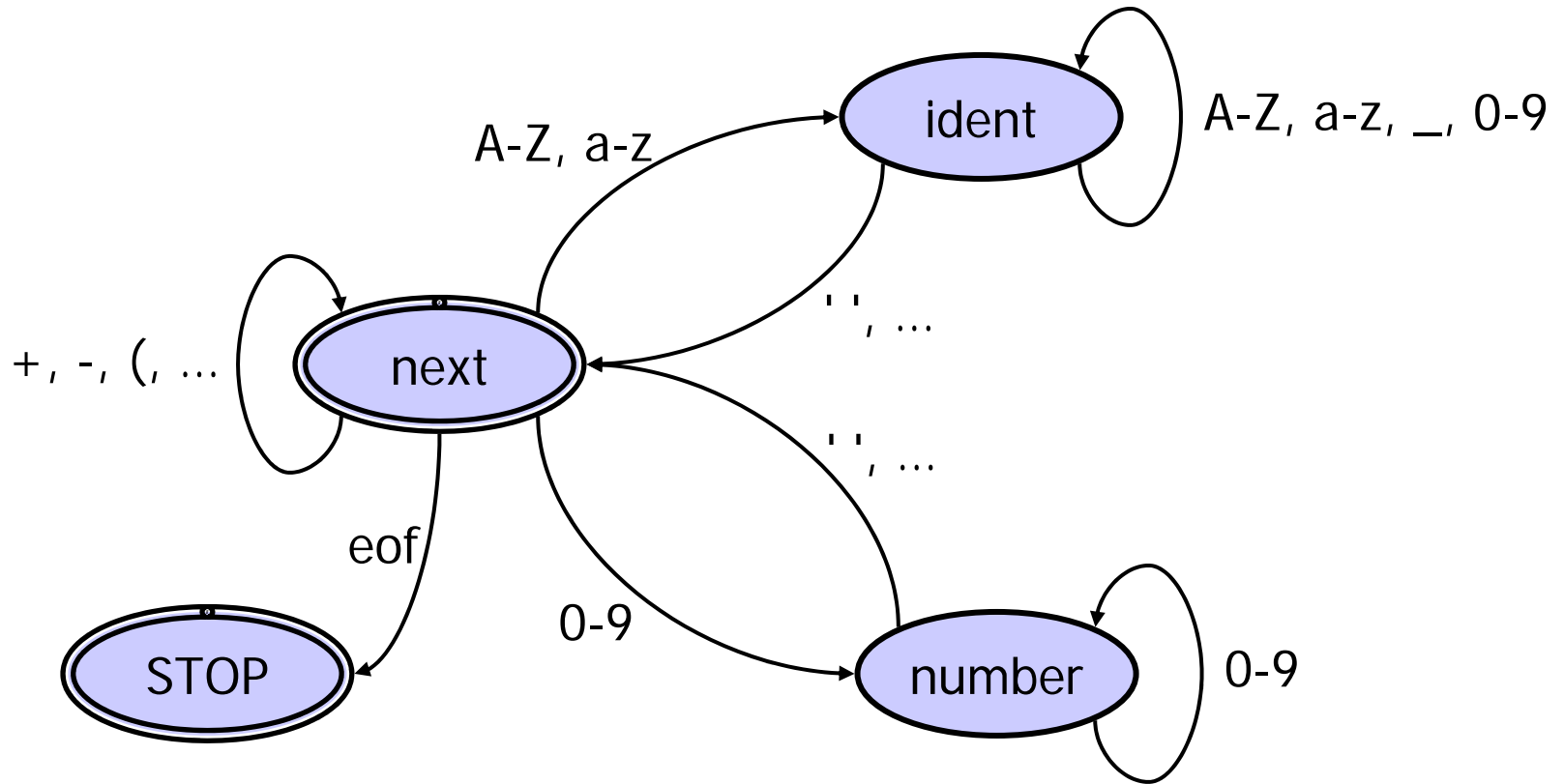
# Scanner.next() (3)

```

...
//----- simple tokens
case ';': t.kind = Token.semicolon; nextCh(); break;    ...
case eofCh: t.kind = Token.endFile; /* no nextCh() ! */ break;
//----- compound tokens
case '=':
    nextCh();
    if (ch == '=') { t.kind = Token.eq; nextCh(); }
    else t.kind = Token.assign;
    break;
case '/':
    nextCh();
    if (ch == '*') { skipComment(); t = next(); /* recursion! */ }
    else t.kind = Token.slash;
    break;
...
}
if (t.kind = Token.none) error("invalid symbol");
return t;
}

```

# Scanner als endlicher Automat





- eine einzige statische Klasse
- Hauptmethode `void main ()` (kein Rückgabewert, keine Parameter)
- Typen: `int` (4 Byte), `char` (1 Byte, ASCII)
- globale und lokale Variablen, globale Konstanten
- eindimensionale Arrays
- Records (sehen aus wie innere Klassen)
- Methodenaufrufe sind *call-by-value* (Variablen sind aber Referenzen)
- Ein-/Ausgabe mit Hilfe der *read-* und *print-*Anweisung
- eingebaute Methoden `ord()`, `chr()`, `len()` und Konstanten `null`
- keine Module, Importanweisungen, ...
- kein GC und auch kein `delete` (Objekte bleiben übrig – who cares ☺ )
- keine `for`-Schleife, nur `while-` und `do-while-`Schleifen
- keine Ausnahmebehandlung (*exception handling*)
- keine Zeiger

# UE 2: Lexikalische Analyse (*Scanner*)



- Testen
  - siehe Testanleitung auf Homepage!
  - nächste Woche in der UE:  
**"Testen mit dem JUnit-Framework"** 😊
- UB-UE2-Angabe.zip:
  - Compilerklassen (Token.java, Messages.java, Gerüst von Scanner.java, ...)
  - Testklassen (AllTests.java, TokenTest.java, ScannerTest.java, ...)
- Abgabe
  - siehe Abgabeanleitung auf Homepage!
  - bis **Mi, 20.10.2004, 12:00**: Aufgabe 2, elektronisch
  - bis Do, 28.10.2004, 8:15: Aufgabe 1, auf Papier & elektronisch