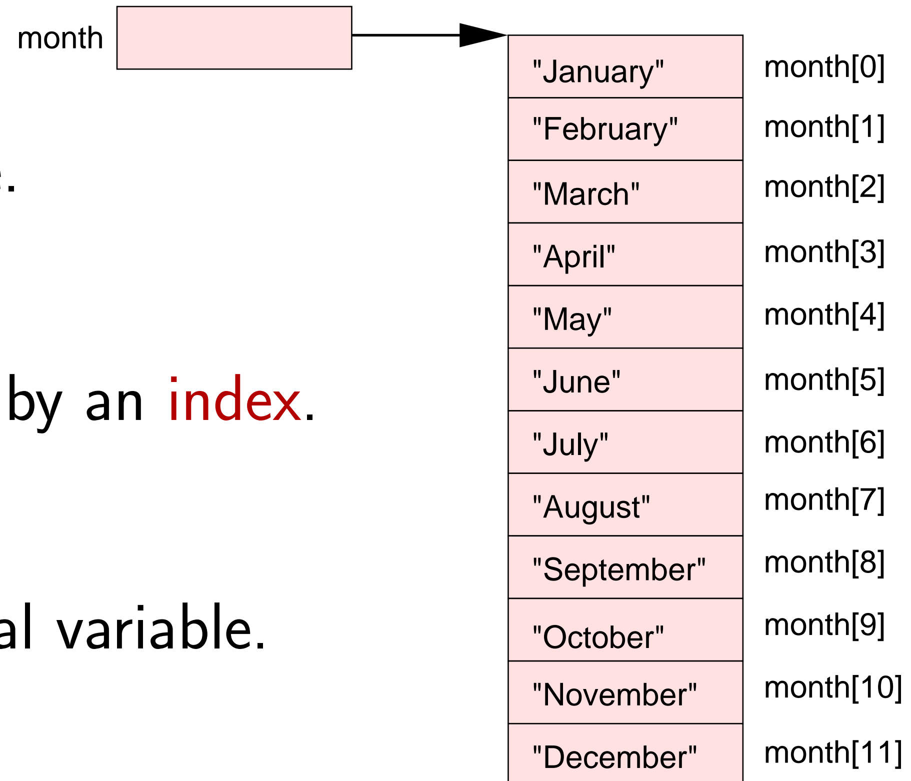
Arrays

Wolfgang Schreiner
Research Institute for Symbolic Computation (RISC)
Johannes Kepler University, Linz, Austria

Wolfgang.Schreiner@risc.jku.at
http://www.risc.jku.at

Arrays

An array is a table of values.



- All values are of the same type.
  - A "String array" `month`
  - `String[] month`
- Each array position is denoted by an index.
  - `month[index]`
  - `0 <= index < month.length`
- Each array entry is an individual variable.
  - `... = ... month[index] ...`
  - `month[index] = ...`

Most important data structure for collecting entities.

Arrays in Java

A Java array is represented by a pointer.

- An array variable holds a 32 bit value.
  - The variable references the array.
  - The variable does not hold the array.
- Assignments are pointer assignments.
  - `a1 = a2`: both variables refer to the same array.
- Comparisons are pointer comparisons.
  - `a1 == a2`: result is true only if the pointers are identical.
- Array variables can be assigned the special value `null`.
  - `null` is the default value for all array types.

Java arrays behave similar to Java objects.

Array Declaration and Creation

```
type[] name
type name[]
```

- Declaration of an array variable/constant *name*
  - *type* is the base type of the array.
  - Only creates the array variable (not the array itself).

```
new type[length]
```

- Creates an array whose base type is *type*.
  - The array has *length* elements.
  - Each element is initialized to the default value of *type*.

Declaration of an array variable is different from creation of an array.

Example

```
String[] month = new String[12];
String[] strings;
strings = new String[10];
strings = month;
month = null;
```

1. Declararation of array variable and creation of array.
2. Only declaration of array variable.
3. Creation of array and assignment to array variable.
4. Assignment of array variables.
5. Assignment of array variable to `null`.

Array Initialization and Assignment

- Whole arrays can be created and initialized in declaration.

type`[]` *name* `=` {*expressions*}`;`

```
String[] month = {"January", "February", ..., "December"};
```

- Array variables can be assigned new values later.

name `= new` *type*`[]` {*expressions*}`;`

```
month = new String[] {"January", "February", "March"};
```

- Size of array and elements are determined by *expressions*.

Especially useful for small constant arrays.

Array Referencing

$array\,[index]$

- *array* is an array whose base type is some *type*.
  - The array has some *length*.
- *index* is an int value.
  - $0 \leq index < length$.
- *array*[*index*] is a variable whose type is *type*.

Basic mechanism for access to array contents.

Array Length

```
array.length
```

- Number of elements in the array referenced by array.
- If array is `null`, a runtime error is triggered.

Useful for arrays that are passed as parameters to methods.

Example

```
String[] month = new String[12];

month[0] = "January";
month[1] = "February";
...
month[11] = "December";

System.out.print("Enter month (1-12): ");
int i = Input.readInt();
if (Input.isOkay() && 1 <= i && i <= month.length)
  System.out.println("Month " + i + ": " + month[i-1]);
```

Example

```
String[] month = new String[12];
System.out.println(month[12]);
```

```
Exception in thread "main"
java.lang.ArrayIndexOutOfBoundsException
        at Main.main(Main.java:6)
```

Bounds of array are automatically checked on every access.

Example: Command Line Arguments

Program arguments are passed as a string array to `main`.

```
class Main
{
  public static void main(String[] args)
  {
    for (int i = 0; i < args.length; i++)
      System.out.println(args[i]);
  }
}
```

```
java Main -option input 2
-option
input
2
```

Iterating over Arrays

Java supports a special form of array iterations:

- Typically the index is used to access the array elements:

```
for (int i = 0; i < a.length; i++)
{
  ... process a[i] ...
}
```

- If index is not needed, an alternative syntax is possible:

```
for (T e : a)
{
  ... process e ...
}
```

Alternative syntax focuses on array elements rather than indices.

Example: Printing an Array

- Definition:

```
public static void printArray(int[] a)
{
  for (int e : a)
    System.out.println(e);
}
```

- Call:

```
int[] arr = { 2, 3, 4 };
printArray(arr);
```

- Output:

```
2
3
4
```

Example: Searching in an Array

```
// i = search(a, x)
// i is the index of the first occurrence of element x in array a
//
// Input condition: a is not null
// Output condition:
//   i == -1 or 0 <= i < a.length
//   if i == -1, then a[j] != x for 0 <= j < a.length
//   otherwise, a[i] = x and a[j] != x for 0 <= j < i
public static int search(int[] a, int x)
{
  final int n = a.length;
  for (int i = 0; i < n; i++)
    if (a[i] == x) return i;
  return -1;
}
```

Example: Prime Number Computation

Write a program which prints all prime numbers less than a given number n.

- Maintain an array *p* of primes computed so far.
  - *p* is initially empty and gradually extended.
- Test every candidate *c* whether it is divided by any element of *p*.
  - If yes, *c* is dropped.
  - If no, *c* is added to *p*.
- Optimization: only test odd $c \geq 3$.
  - 2 is the only even prime.

Need a table to keep track of progress.

Main Method

```
public static void printPrimes(int n)
{
  if (n <= 2) return;              // no prime less than 2
  System.out.println(2);           // 2 is prime

  int len = 1000;                  // initial size of p
  int[] p = new int[len];          // prime table
  int l = 0;                       // number of elements in p

  for (int c = 3; c < n; c += 2) // check odd candidates c
  { if (!isPrime(c, p, l)) continue;
    System.out.println(c);         // c is prime
    if (l == len)                  // table is full
    { len = 2*len;                 // double size of table
      p = resize(p, len);
    }
    p[l] = c;                      // enter c into table
    l = l+1;
  }
}
```

Auxiliary Methods

```
public static boolean isPrime(int c, int[] p, int l)
{
  for (int i = 0; i < l; i++)
    if (c % p[i] == 0) return false;
  return true;
}

public static int[] resize(int[] p, int len)
{
  int l = p.length;
  int[] q = new int[len];
  for (int i = 0; i < l; i++)
    q[i] = p[i];
  return q;
}
```

Objects containing Arrays

An array alone is usually not sufficient to represent program data.

- Data structure consists of array and additional data (e.g. indices).
- All pieces can be packed into a single object.
- Object provides via methods access to the data structure.

Arrays are frequently part of an object representation.

Example: Prime Number Computation

```
public static void printPrimes(int n)
{ if (n <= 2) return;           // no prime less than 2
  System.out.println(2);         // 2 is prime

  Table p = new Table();         // prime table
  for (int c = 3; c < n; c += 2) // check odd candidates c
  {
    if (!isPrime(c, p)) continue;
    System.out.println(c);       // c is prime
    p.add(c);                    // add c to p
  }
}

public static boolean isPrime(int c, Table p)
{ final int l = p.getNumber();
  for (int i = 0; i < l; i++)
    if (c % p.getElement(i) == 0) return false;
  return true;
}
```

Data Structure

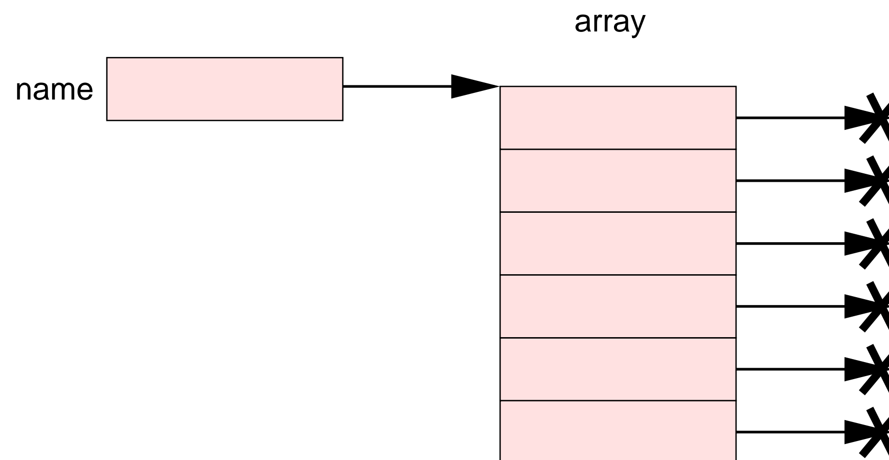
```
public class Table
{ public int[] p = new int[1000]; // the table data
  public int l = 0;               // number of elements in p

  public void add(int e)
  { if (l == p.length) resize();
    p[l] = e;
    l = l+1;
  }
  public void resize()
  { int l = p.length;
    int[] q = new int[2*l];
    for (int i = 0; i < l; i++)
      q[i] = p[i];
    p = q;
  }
  public int getNumber() { return l; }
  public int getElement(int i) { return p[i]; }
}
```

Arrays Containing Objects

An array may also contain objects.

```
Class[] name = new Class[length]
```

- Declares array variable *name* whose base type is *Class*.
- Assigns to *name* new array with *length* elements.
  - Array elements are initialized to the default value `null`.

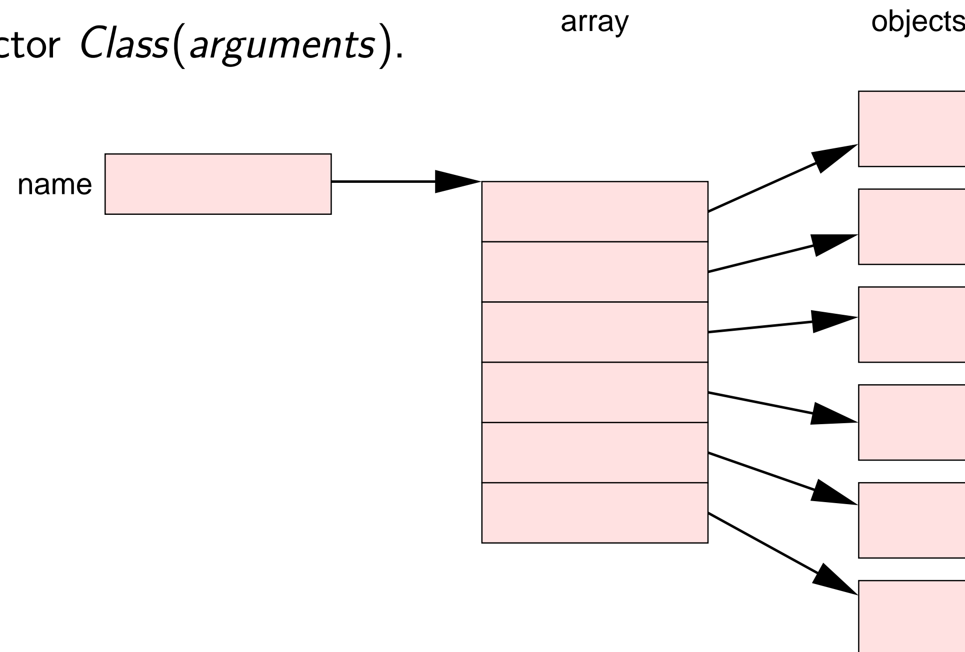


Array Initialization

The array elements have to be initialized.

```
for (int i = 0; i < length; i++)
  name[i] = new Class(arguments)
```

- A new object is allocated for each position in the array.
  - Call of constructor *Class*(*arguments*).



Example: Phone Book

Write a program that reads a list of phone book entries, i.e., pairs of a name and a phone number. The program then reads sequences of names and prints the corresponding phone number.

```
public static void main(String[] args)
{
  Entry[] book = readPhoneBook();
  usePhoneBook(book);
}
```

Central data structure is an array of objects.

A Phone Book Entry

```
public class Entry
{
  public String name;
  public String number;

  public Entry(String name, String number)
  { this.name = name;
    this.number = number;
  }

  public String getName() { return name; }
  public String getNumber() { return number; }
}
```

Reading a Phone Book

```
public static Entry[] readPhoneBook()
{
  System.out.print("How many entries? ");
  int n = Input.readInt();
  Entry[] book = new Entry[n];
  for (int i=0; i<n; i++)
  {
    System.out.println("Entry " + (i+1) + ":");
    book[i] = readEntry();
  }
  return book;
}
```

Using a Phone Book

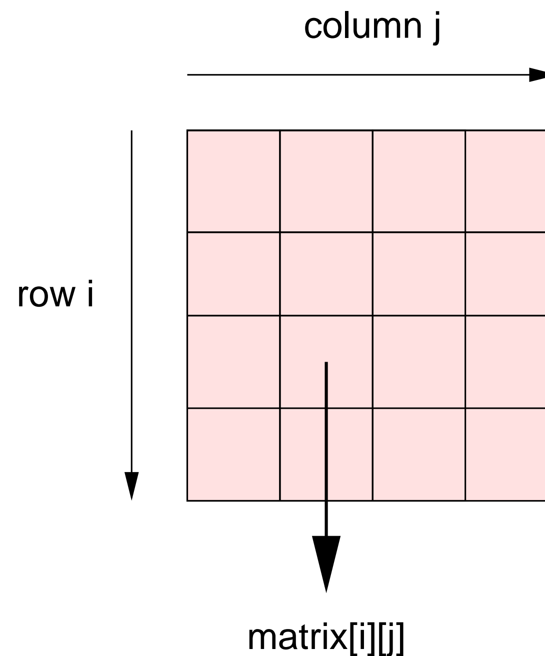
```
public static void usePhoneBook(Entry[] book)
{
  while (true)
  {
    System.out.print("Another lookup (y/n)? ");
    char ch = Input.readChar();
    if (ch == 'n') return;
    System.out.print("Name: ");
    String name = Input.readString();
    Entry entry = search(book, name);
    if (entry == null)
      System.out.println("Name not found.");
    else
      System.out.println("Number: " + entry.getNumber());
  }
}
```

Auxiliary Methods

```
public static Entry readEntry()
{
  System.out.print("Enter name: ");
  String name = Input.readString();
  System.out.print("Enter number: ");
  String number = Input.readString();
  return new Entry(name, number);
}

public static Entry search(Entry[] book, String name)
{
  final int n = book.length;
  for(int i = 0; i < n; i++)
    if (book[i].getName().equals(name)) return book[i];
  return null;
}
```

Multi-dimensional Arrays

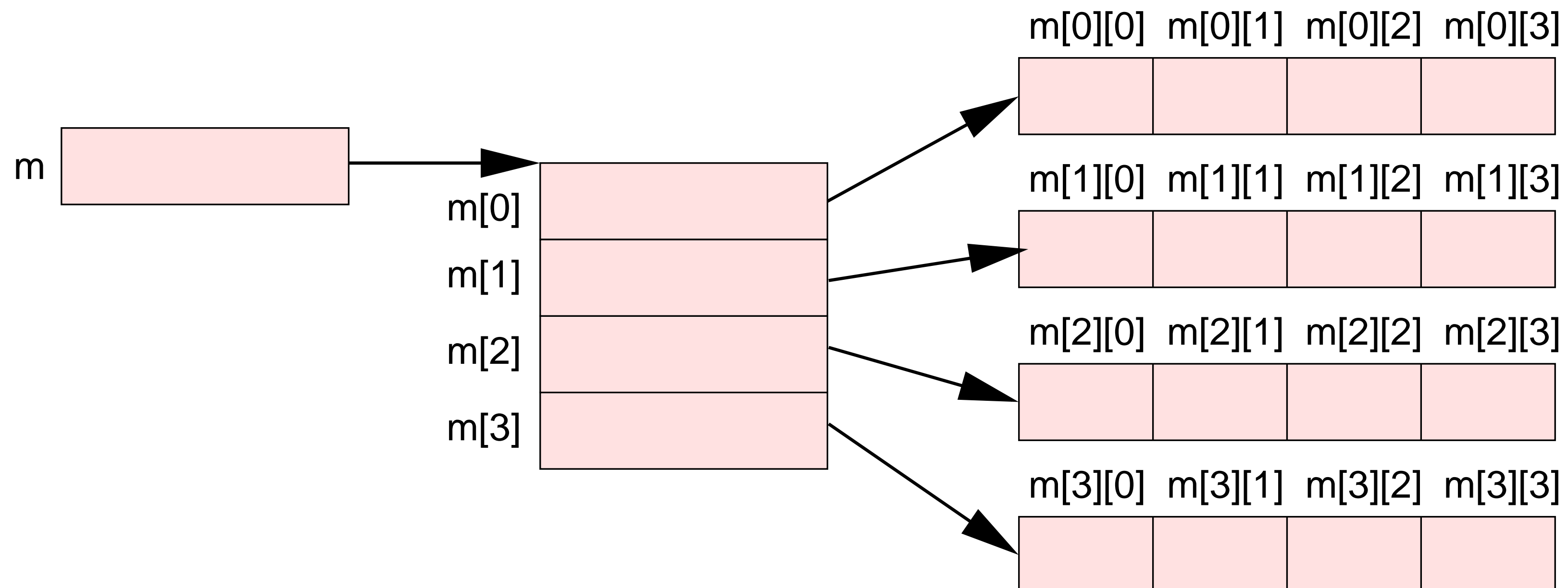
How can we implement a matrix of values?



A matrix element is referenced by a pair of indices.

Java Representation

A matrix is an array of matrix rows (i.e. arrays).



One-dimensional arrays can simulate multi-dimensional arrays.

Declaration and Creation

```
type[][] name
```

- Declares a matrix variable *name*.
  - Array of (array of *type*).

```
new type[rows][columns];
```

- Allocates a matrix whose base type is *type*.
  - *rows* is the number of rows in matrix.
  - *columns* is the number of columns in each row.

Declaration of a matrix variable is different from allocation of a matrix.

Matrix Initialization

- Whole matrices can be initialized in the declaration.

```
type[][] name = { {value, ...}, ... }

int[][] m = { {1, 2, 3}, {4, 5, 6} } // m[1][2] == 6
```

- Matrix variable can be later assigned a new value.

```
name = new type[][]  { {value, ...}, ... };

m = new int[][] { {1, 2}, {3, 4}, {5, 6} }; // m[2][1] == 6
```

- Size of matrix and elements are determined by $\{$*value*$, \ldots\}, \ldots$

Especially useful for small constant matrices.

Matrix Referencing

How can we work with matrices?

- Element access:

  name [*row*] [*column*]

- Number of rows:

  name.length

- Number of columns:

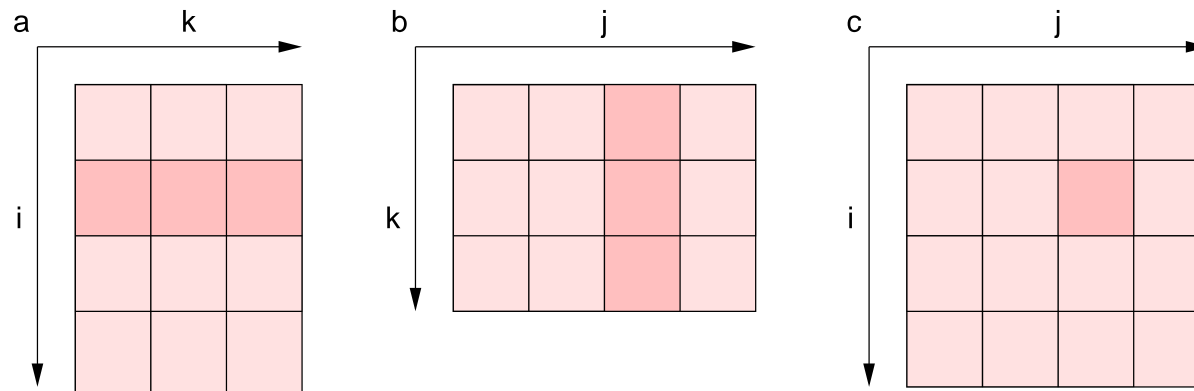
  name [0] .length

Use array mechanisms.

Example: Matrix Multiplication

Multiply two matrices a and b giving a result matrix c.

$$c[i][j] = \sum_{0 \leq k < n} a[i][k] * b[k][j]$$



Method

```
public static float[][] matMult(float[][] a, float[][] b)
{ final int M = a.length;    // a is M x Q
  final int N = b[0].length; // b is Q x N
  final int Q = a[0].length; // c is M x N

  float[][] c = new float[M][N];
  for (int i = 0; i < M; i++)
  { for (int j = 0; j < N; j++)
    { double sum = 0;
      for (int k = 0; k < Q; k++)
        sum = sum + a[i][k] * b[k][j];
      c[i][j] = (float)sum;
    }
  }
  return c;
}
```

The Class `Vector<T>`

`java.util.Vector` provides dynamically growing sequences.

`Vector<T>()`

- This constructor creates a vector that is initially empty.

`void add(T object)`

- This object method adds the specified *object* to the end of *this* vector.

`T get(int index)`

- This object method returns the object at the specified *index*.

`int size()`

- This object method returns the number of elements in *this* vector.

Elements of the generic class `Vector<T>` have object type T.

Example

We can declare a vector without giving a specific size.

```
Vector<Integer> v = new Vector<Integer>();
v.add(0);                    // autoboxing int -> Integer
v.add(1);
v.add(2);
int i = v.get(1);            // autounboxing Integer -> int
System.out.println(i);
```

Type T has to be an object type, but by autoboxing also vector elements of atomic type can be used.