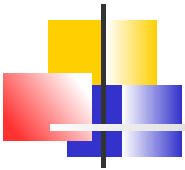
Software and Programming I

Classes and Arrays

Roman Kontchakov / Carsten Fuhs

Birkbeck, University of London



Outline

- Class `Object`
  - Section 9.5
- Arrays
- Common Array Algorithms
  - Sections 6.1–6.4
- slides are available at
  `www.dcs.bbk.ac.uk/~roman/sp1`

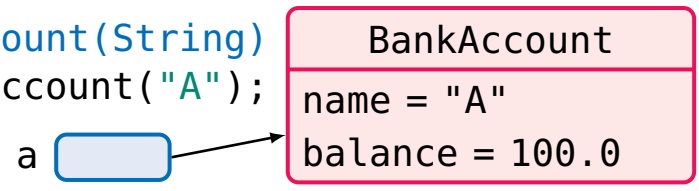
Overloading

Methods (and constructors) can have the same name
(provided their signatures (i.e., name and parameter types) are different)

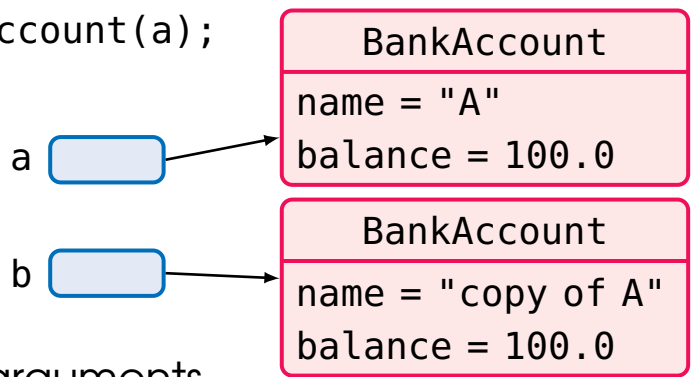
```
1  public class BankAccount {
2      // ...
3      public BankAccount(String name) {
4          this.name = name;
5          this.balance = 0;
6      }
7      public BankAccount(BankAccount a) {
8          this.name = "copy of " + a.name;
9          this.balance = a.balance;
10     }
11 }
```

Overloading (2)

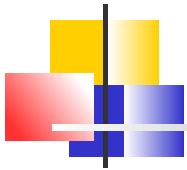
```
1 // constructor 1: BankAccount(String)
2 BankAccount a = new BankAccount("A");
3 a.deposit(100);
```




```
4 // constructor 2: BankAccount(BankAccount)
5 BankAccount b = new BankAccount(a);
```



NB: compile-time types of arguments determine which constructor is called



Overriding, Inheritance and Polymorphism

a subclass **inherits** all methods that it does not **override**

a subclass method overrides a public method from a superclass
if both methods have the **same signature**

a subclass can override a superclass method by providing a new **implementation**

polymorphism:

- the type of the **reference** determines which **method signatures** we may call
  (checked at compile-time)
- the type of the **actual object** determines which **method implementation** is invoked (at run-time)

Overriding, Inheritance and Polymorphism: Example (1)

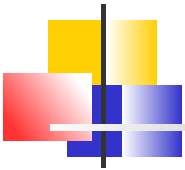
```
1  public class A {
2      public int f() { return 1; }
3      public int f(int i) { return 5; }
4      public int g() { return 2; }
5  }
6  public class B extends A {// g(), f(int) are inherited
7      public int f() { return 3; } // f() is overridden
8      public int f(String s) { return 6; } // f(String)
9                               // DOES NOT override f(int)
10     public int h() { return 4; } // h() is new
11 }
1  A a = new A();                    // prints 152
2  System.out.println(a.f() + "" + a.f(1) + "" + a.g());
3  // a.h() or a.f("1") is a compile-time error
```

Overriding, Inheritance and Polymorphism: Example (2)

```
1  public class A {
2      public int f() { return 1; }
3      public int f(int i) { return 5; }
4      public int g() { return 2; }
5  }
6  public class B extends A {// g(), f(int) are inherited
7      public int f() { return 3; } // f() is overridden
8      public int f(String s) { return 6; } // f(String)
9                               // DOES not override f(int)
10     public int h() { return 4; } // h() is new
11 }
1  B b = new B();          // prints 35624
2  System.out.println(b.f() + "" + b.f(1) + "" + b.f("1")
3                             + "" + b.g() + "" + b.h());
```

Overriding, Inheritance and Polymorphism: Example (3)

```
1  public class A {
2      public int f() { return 1; }
3      public int f(int i) { return 5; }
4      public int g() { return 2; }
5  }
6  public class B extends A {// g(), f(int) are inherited
7      public int f() { return 3; } // f() is overridden
8      public int f(String s) { return 6; } // f(String)
9                                 // DOES not override f(int)
10     public int h() { return 4; } // h() is new
11 }
1  A c = new B();// c provides A's methods, uses B's code
2  System.out.println(c.f()+""+c.f(1)+""+c.g()); // 352
3  // c.h() or c.f("1") is a compile-time error
```

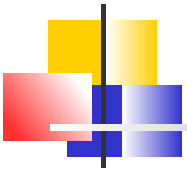


The Object Class

every class declared without the explicit `extends` clause automatically extends the class `Object`

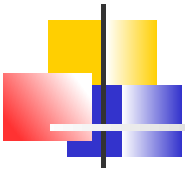
class `Object` has method `String toString()`, which one can **override**

```
1 public class BankAccount {
2     ...
3     public String toString() {
4         return "account " + name +
5                 ", balance = " + balance;
6     }
7 }
```



Overriding toString()

```
1  public class CurrentAccount extends BankAccount {
2      ...
3      public String toString() {
4          return "current " + super.toString();
5      }
6  }
7  public class SavingsAccount extends BankAccount {
8      ...
9      public String toString() {
10         return "savings " + super.toString() +
11                 ", interest rate = " + interestRate;
12     }
13 }
```



Using Object.toString()

the method `toString()` is called, e.g., in `System.out.println`

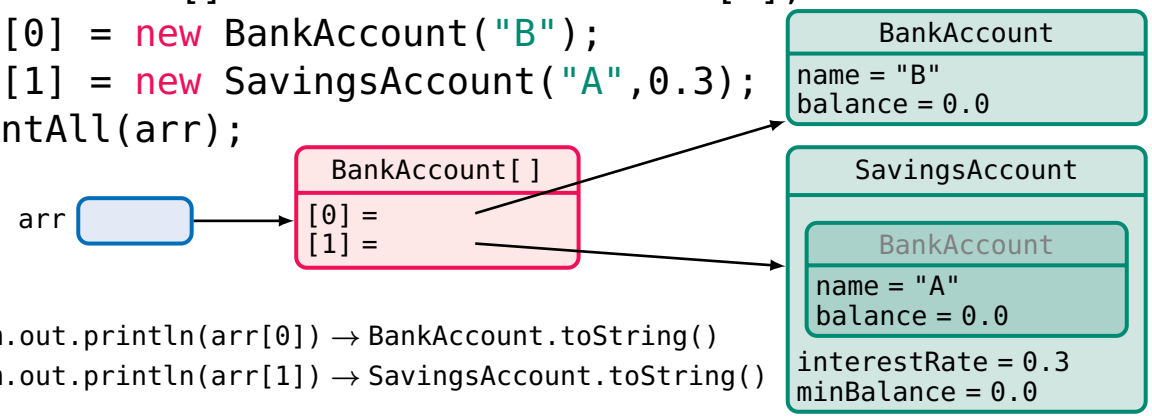
```
1 public static void printAll(BankAccount[] accounts) {
2     for (BankAccount a: accounts)
3         System.out.println(a);
4 }
```

the implementation of method `println(Object obj)`
invokes `obj.toString()`

which `toString()` implementation is invoked
depends on the **run-time type** of `obj` (**polymorphism**)

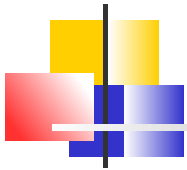
Polymorphism in Action

```
1 BankAccount[] arr = new BankAccount[2];
2 arr[0] = new BankAccount("B");
3 arr[1] = new SavingsAccount("A",0.3);
4 printAll(arr);
```



`System.out.println(arr[0])` → `BankAccount.toString()`
`System.out.println(arr[1])` → `SavingsAccount.toString()`

```
account B, balance = 0.0
savings account A, balance = 0.0, interest rate = 0.3
```



More Methods of Object

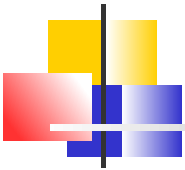
- `boolean equals(Object obj)`
  indicates whether some other object is "equal to" this one

? would a method `boolean equals(BankAccount obj)` in class `BankAccount` be useful?

- `int hashCode()`
  returns a hash code value for the object

  (used in collections)

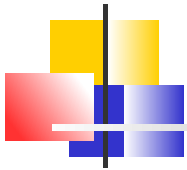
- `Class getClass()`
  returns the run-time **class** of an object



Nested Loops

```
1 for (int r = 8; r >= 1; r--) {
2     for (char c = 'a'; c <= 'h'; c++)  // char is
3                                        // an integral datatype
4         System.out.print(" " + c + r + " ");
5     System.out.println();
6 }
```

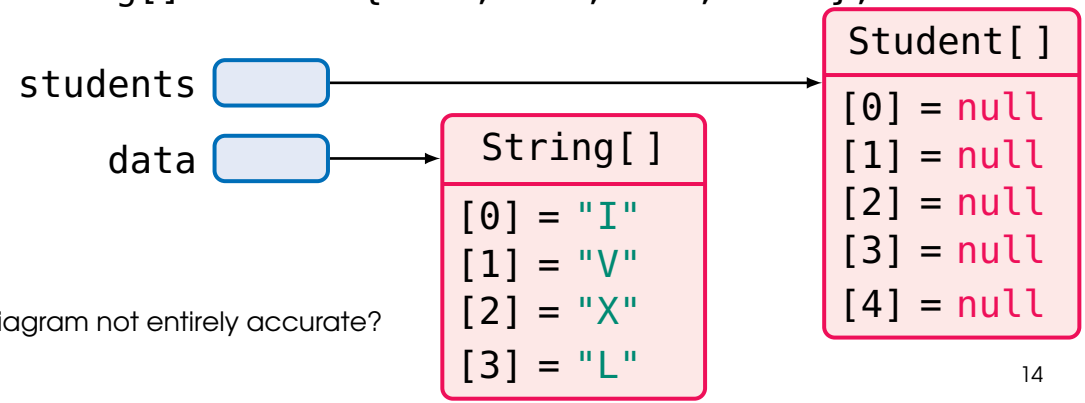
```
a8 b8 c8 d8 e8 f8 g8 h8
a7 b7 c7 d7 e7 f7 g7 h7
a6 b6 c6 d6 e6 f6 g6 h6
a5 b5 c5 d5 e5 f5 g5 h5
a4 b4 c4 d4 e4 f4 g4 h4
a3 b3 c3 d3 e3 f3 g3 h3
a2 b2 c2 d2 e2 f2 g2 h2
a1 b1 c1 d1 e1 f1 g1 h1
```



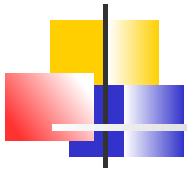
Arrays

- an **array** collects a **sequence** of values of the same **type**

```
1 // empty array of 5 students
2 Student[] students = new Student[5];
3 // list of initial values
4 String[] data = { "I", "V", "X", "L" };
```



NB: why is this diagram not entirely accurate?



Array Elements

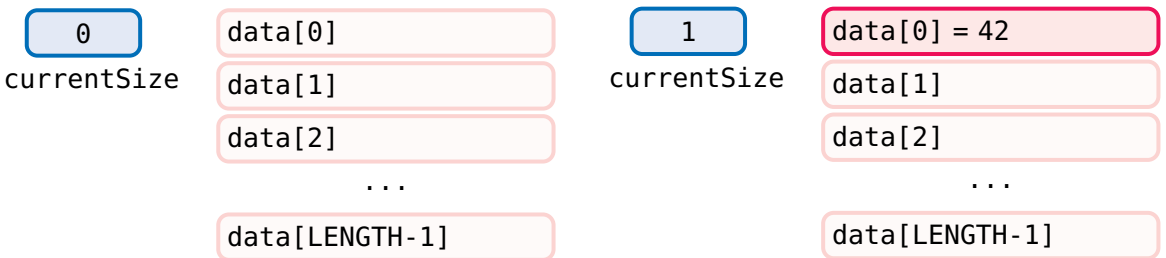
- individual elements in an array `data` are accessed by an integer index `i`, using the notation `data[i]`
- an array element can be used in expressions like any other variable
- the elements of arrays are numbered starting at **0**
- use the expression `data.length` to find the **number** of elements in an array `data`

```
1 int[] data = { 2, 3, 5, 7, 11 };
2 for (int i = 0; i < data.length / 2; i++)
3     data[data.length - 1 - i] = data[i];
```

The Length of Arrays is Fixed

come up with a guess on the maximum number of elements and keep a companion variable for the **current size**

```
1 final int LENGTH = 100; // max number of elements
2 // partially filled array
3 double[] data = new double[LENGTH];
4 int currentSize = 0; // the actual number of elements
5 data[currentSize] = 42; // insert 42
6 currentSize++; // increase the number of elements
```

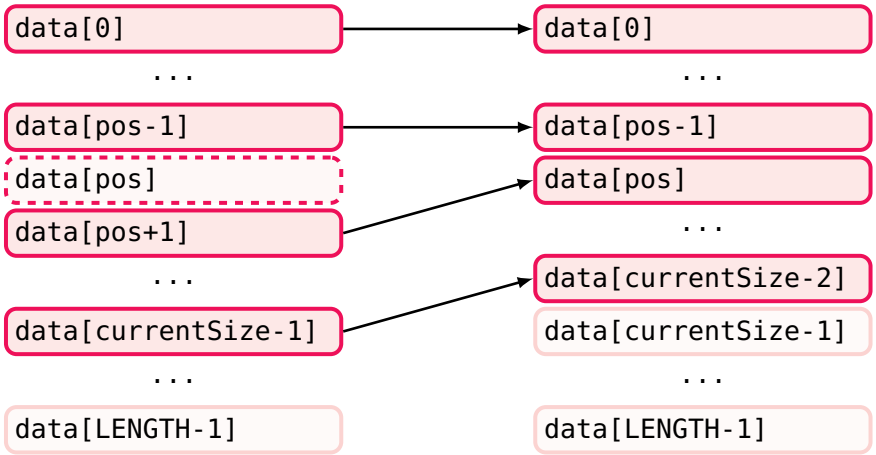


Partially Filled Arrays

```
1   Scanner in = new Scanner(System.in);
2   while (in.hasNextDouble())  // read all doubles
3       if (currentSize < data.length) {
4           // currentSize is the first position
5           //                      available
6           data[currentSize] = in.nextDouble();
7           // update the actual number of elements
8           currentSize++;
9       }
10  // the actual elements are indexed
11  //                from 0  to currentSize-1
12  for (int i = 0; i < currentSize; i++)
13      System.out.println(data[i]);
```

Partially Filled Arrays: Removing an Element

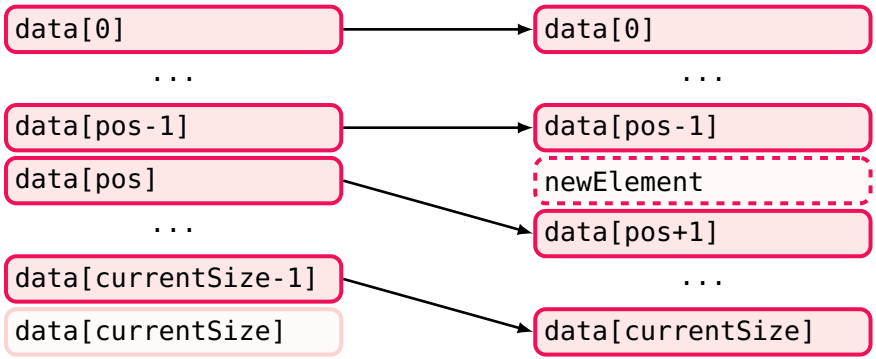
removing the element at position `pos`




```
1 for (int i = pos; i < currentSize - 1; i++)
2     data[i] = data[i+1];
3 currentSize--; // update the actual number of elements
```

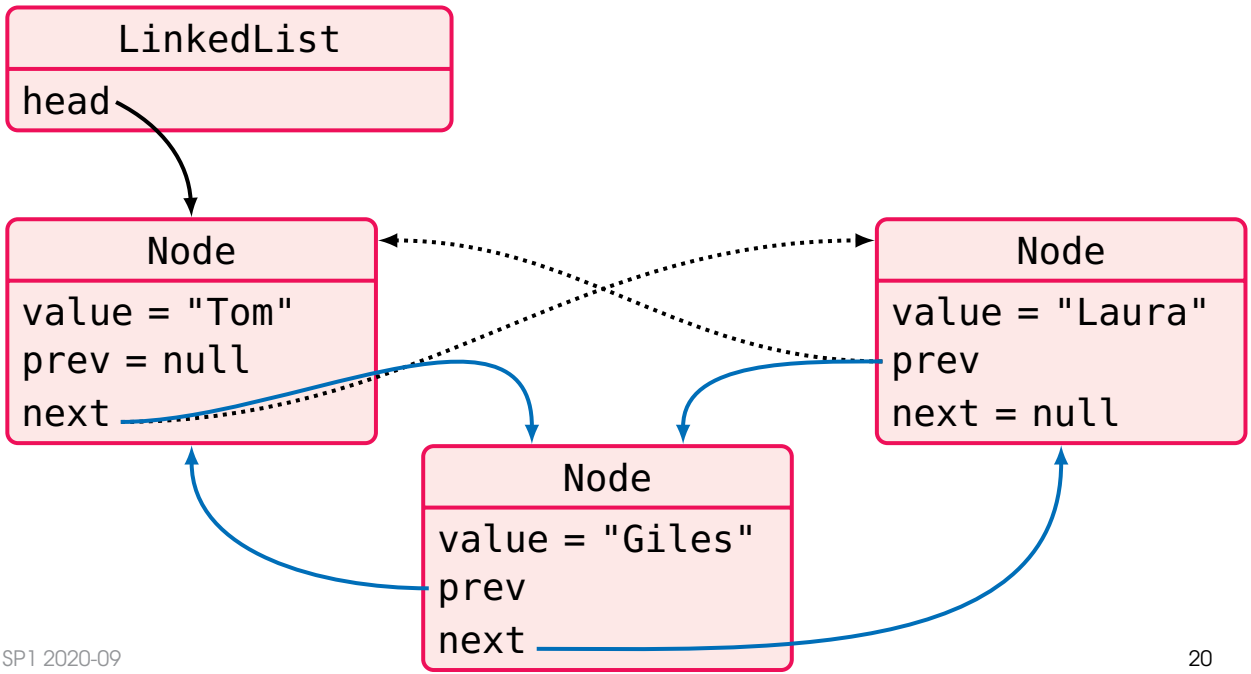
Partially Filled Arrays: Inserting an Element

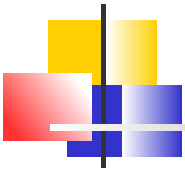
inserting
a newElement
at position
pos




```
1 if (currentSize < data.length) { // check space
2     for (int i = currentSize; i > pos; i--)
3         data[i] = data[i-1];
4     data[pos] = newElement; // place into array
5     currentSize++; // update the number of elements
6 }
```

LinkedList: Inserting Elements

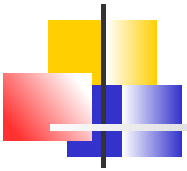




LinkedList: Inserting Elements

```
1  public class Node {
2      private String value;
3      private Node prev, next;
4
5      public Node(Node current, String value) {
6          this.value = value;
7          this.next = current.next;
8          this.prev = current;
9          if (current.next != null)
10             current.next.prev = this;
11         current.next = this;
12     }
13 }
```

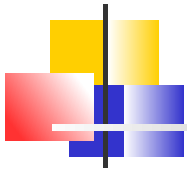
NB: effective insertion and deletions
indexing, however, is slow:
finding the nth element takes n steps



Bubble Sort: Idea

Repeatedly step through the list to be sorted,
comparing each pair of adjacent items and
swapping them if they are in the wrong order.

The pass through the list is repeated
until no swaps are needed,
which indicates that the list is **sorted**



Bubble Sort: Example

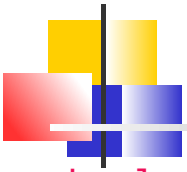
```
pass 1: ( 5 1 4 2 8 ) → ( 1 5 4 2 8 ), swap
        ( 1 5 4 2 8 ) → ( 1 4 5 2 8 ), swap
        ( 1 4 5 2 8 ) → ( 1 4 2 5 8 ), swap
        ( 1 4 2 5 8 ) → ( 1 4 2 5 8 )

pass 2: ( 1 4 2 5 8 ) → ( 1 4 2 5 8 )
        ( 1 4 2 5 8 ) → ( 1 2 4 5 8 ), swap
        ( 1 2 4 5 8 ) → ( 1 2 4 5 8 )
        ( 1 2 4 5 8 ) → ( 1 2 4 5 8 )
```

The array is already sorted,
but the algorithm does not know if it is completed
It needs one whole pass without any swap to know it is sorted

NB: does it need to go until the very end on every pass?

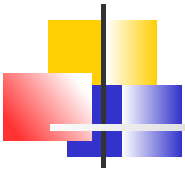
NB: how many steps does the algorithm require?

best sorting algorithms require $\mathcal{O}(n \log n)$ steps



Bubble Sort: Implementation

```
1  boolean swapped;
2  do {
3      swapped = false;
4      // start from 1, not 0!
5      for (int i = 1; i < data.length; i++) {
6          if (data[i-1] > data[i]) {
7              double t = data[i-1];
8              data[i-1] = data[i];
9              data[i] = t;
10             swapped = true;
11         }
12     }
13 } while (swapped);
```



Take Home Messages

- every class automatically extends the class `Object`
- an array index in an array `data` must be
  `>= 0` and `< data.length`
- arrays can occur as method parameters
  and return values (passing references)
- with a partially filled array,
  keep a companion variable for the current size