Arrays and Array Lists

Advanced Programming

ICOM 4015

Lecture 7

Reading: Java Concepts Chapter 8

Lecture Goals

- To become familiar with using arrays and array lists
- To learn about wrapper classes, auto-boxing and the generalized `for` loop
- To study common array algorithms
- To learn how to use two-dimensional arrays
- To understand when to choose array lists and arrays in your programs
- To implement partially filled arrays

Arrays

- **Array: Sequence of values of the same type**
- **Construct array:**

```
new double[10]
```

- **Store in variable of type `double[ ]`**

```
double[] data = new double[10];
```

Continued…

Arrays

- **When array is created, all values are initialized depending on array type:**
  - Numbers: 0
  - Boolean: `false`
  - Object References: `null`

Arrays

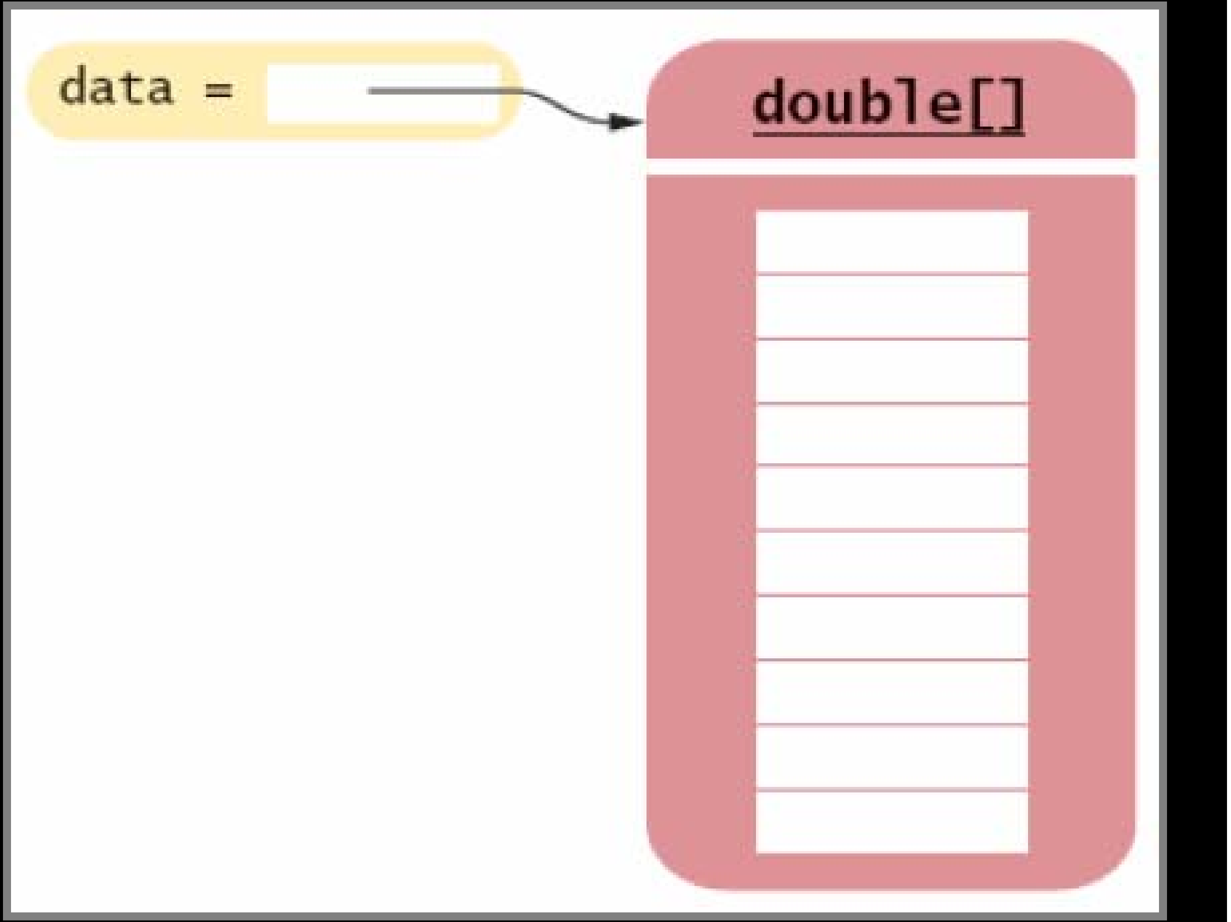


Figure 1:
An Array Reference and an Array

Arrays

- Use [ ] to access an element

```
data[2] = 29.95;
```

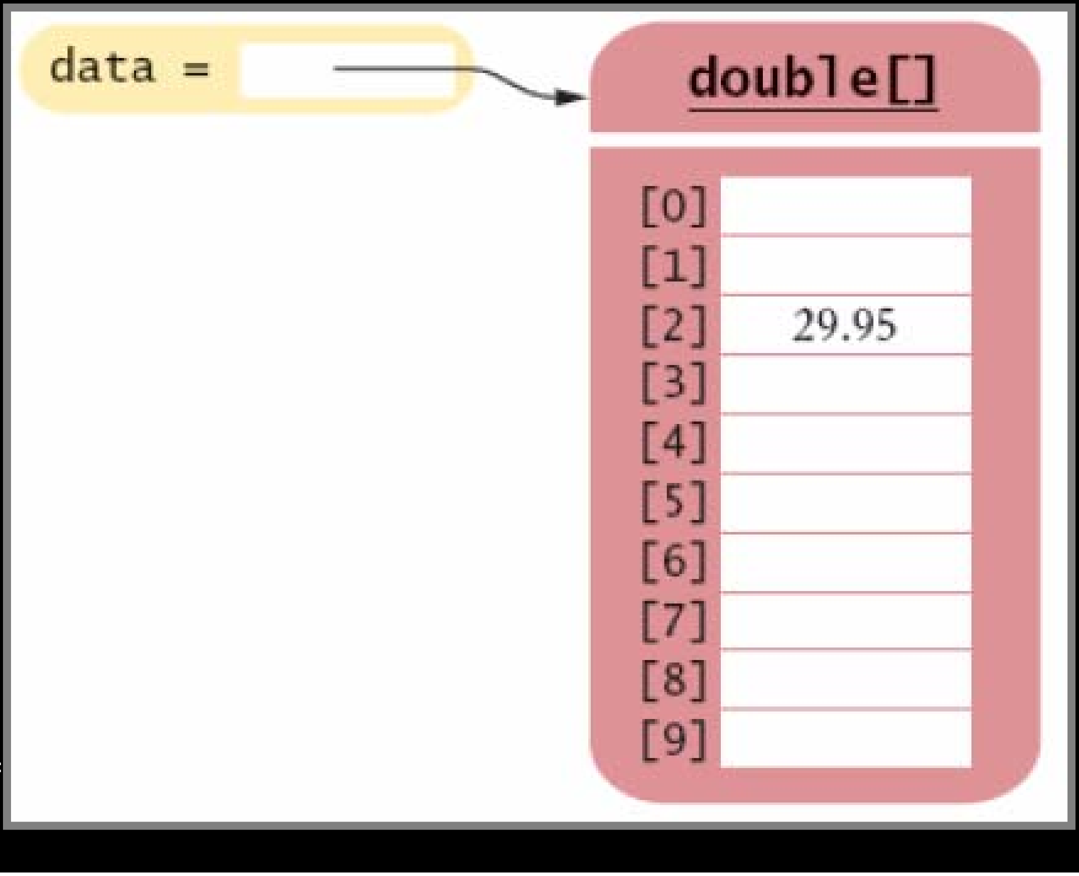


Figure 2:
Storing a Value in an Array

Arrays

- **Using the value stored:**

```
System.out.println("The value of this data item is " + data[4]);
```

- **Get array length as `data.length`. (Not a method!)**
- **Index values range from `0` to `length - 1`**

Continued…

Arrays

- Accessing a nonexistent element results in a bounds error

```
double[] data = new double[10];
data[10] = 29.95; // ERROR
```

- Limitation: Arrays have fixed length

Syntax 8.1: Array Construction

```
new typeName[length]
```

Example:

```
new double[10]
```

Purpose:

To construct an array with a given number of elements

Syntax 8.2: Array Element Access

```
arrayReference[index]
```

Example:

```
 data[2]
```

Purpose:

To access an element in an array

Self Check

1. **What elements does the data array contain after the following statements?**

```
double[] data = new double[10];
for (int i = 0; i < data.length; i++) data[i] = i * i;
```

Self Check

2. What do the following program segments print? Or, if there is an error, describe the error and specify whether it is detected at compile-time or at run-time.

```
1.   double[] a = new double[10];
     System.out.println(a[0]);
2.   double[] b = new double[10];
     System.out.println(b[10]);
3.   double[] c;
     System.out.println(c[0]);
```

Answers

1. **0, 1, 4, 9, 16, 25, 36, 49, 64, 81, but not 100**
2.
    1. 0
    2. a run-time error: array index out of bounds
    3. a compile-time error: c is not initialized

Array Lists

- The `ArrayList` class manages a sequence of objects
- Can grow and shrink as needed
- `ArrayList` class supplies methods for many common tasks, such as inserting and removing elements

Continued…

Array Lists

- The `ArrayList` class is a generic class: `ArrayList<T>` collects objects of type `T`:

```
ArrayList<BankAccount> accounts = new ArrayList<BankAccount>();
accounts.add(new BankAccount(1001));
accounts.add(new BankAccount(1015));
accounts.add(new BankAccount(1022));
```

- `size` method yields number of elements

Retrieving Array List Elements

- Use **`get`** method
- Index starts at 0
-
  ```
  BankAccount anAccount = accounts.get(2);
     // gets the third element of the array list
  ```
- Bounds error if index is out of range

Continued…

Retrieving Array List Elements

- **Most common bounds error:**

```
int i = accounts.size();
anAccount = accounts.get(i); // Error
// legal index values are 0. . .i-1
```

Adding Elements

- `set` overwrites an existing value

```
BankAccount anAccount = new BankAccount(1729);
accounts.set(2, anAccount);
```

- `add` adds a new value before the index

```
accounts.add(i, a)
```

Continued…

Adding Elements

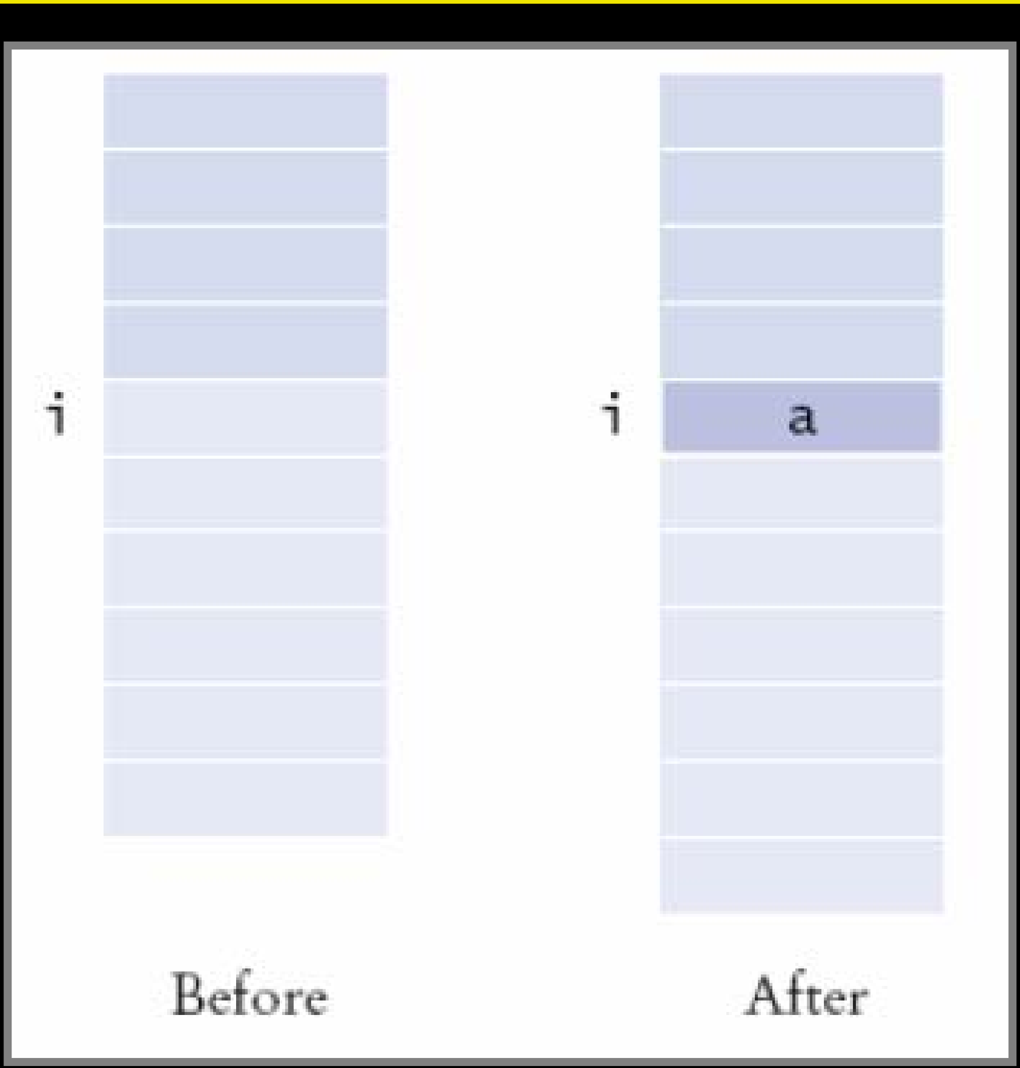


Figure 3:
Adding an Element in the Middle of an Array List

Removing Elements

- `remove` removes an element at an index

```
Accounts.remove(i)
```

Continued…

Removing Elements

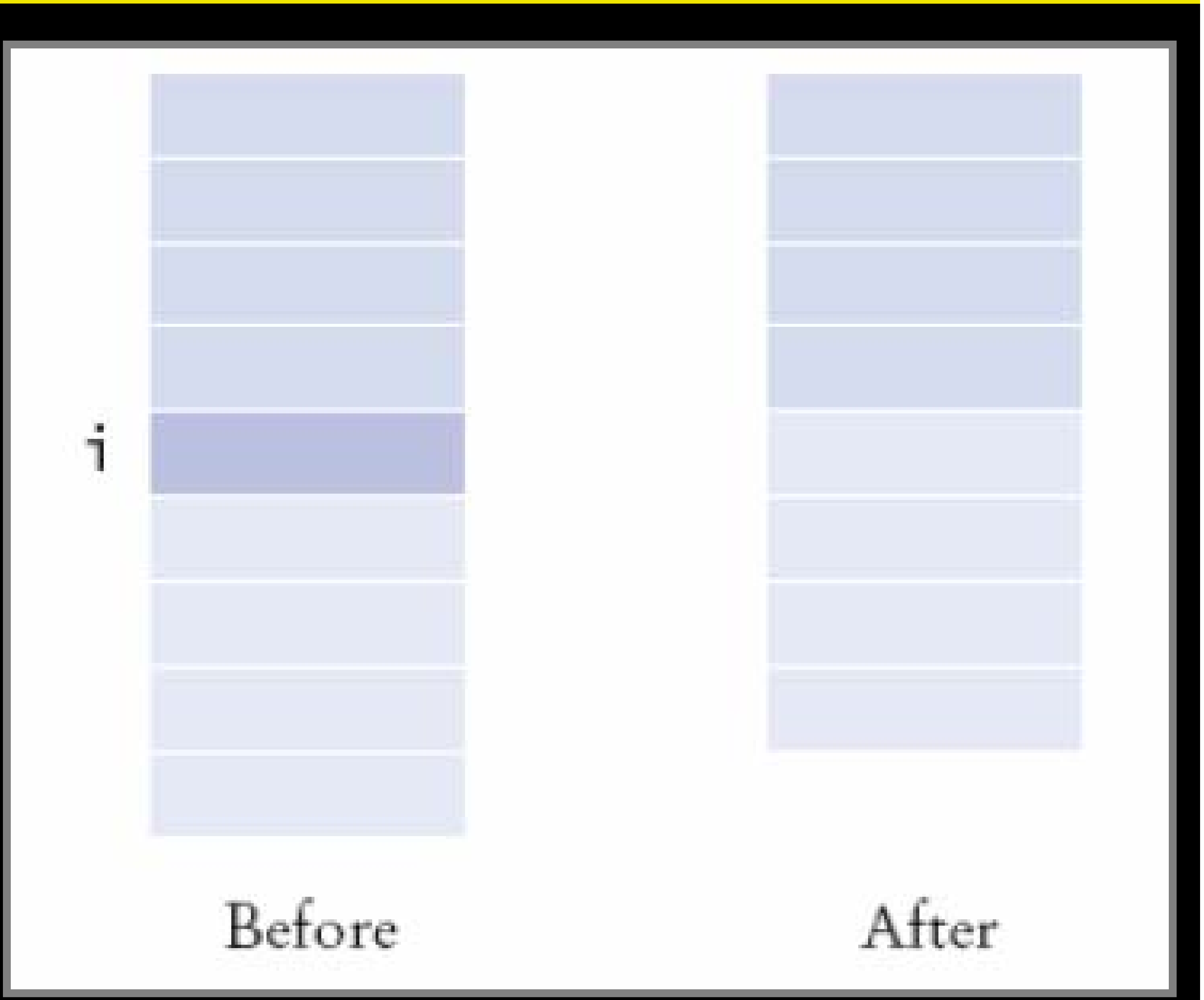


Figure 4:
Removing an Element in the Middle of an Array List

File: ArrayListTester.java

```java
import java.util.ArrayList;

/**
   This program tests the ArrayList class.
*/
public class ArrayListTester
{
   public static void main(String[] args)
   {
      ArrayList<BankAccount> accounts
            = new ArrayList<BankAccount>();
      accounts.add(new BankAccount(1001));
      accounts.add(new BankAccount(1015));
      accounts.add(new BankAccount(1729));
      accounts.add(1, new BankAccount(1008));
      accounts.remove(0);
```

Continued…

File: ArrayListTester.java

```java

      System.out.println("size=" + accounts.size());
      BankAccount first = accounts.get(0);
      System.out.println("first account number="
            + first.getAccountNumber());
      BankAccount last = accounts.get(accounts.size() - 1);
      System.out.println("last account number="
            + last.getAccountNumber());
   }
}
```

File: BankAccount.java

```java
/**
   A bank account has a balance that can be changed by
   deposits and withdrawals.
*/
public class BankAccount
{
   /**
      Constructs a bank account with a zero balance
      @param anAccountNumber the account number for this account
   */
   public BankAccount(int anAccountNumber)
   {
      accountNumber = anAccountNumber;
      balance = 0;
   }

```

Continued…

File: BankAccount.java

```java
   /**
      Constructs a bank account with a given balance
      @param anAccountNumber the account number for this account
      @param initialBalance the initial balance
   */
   public BankAccount(int anAccountNumber, double initialBalance)
   {
      accountNumber = anAccountNumber;
      balance = initialBalance;
   }

   /**
      Gets the account number of this bank account.
      @return the account number
   */
   public int getAccountNumber()
   {
      return accountNumber;
   }
```

Continued…

File: BankAccount.java

```java

    /**
       Deposits money into the bank account.
       @param amount the amount to deposit
    */
    public void deposit(double amount)
    {
       double newBalance = balance + amount;
       balance = newBalance;
    }

    /**
       Withdraws money from the bank account.
       @param amount the amount to withdraw
    */
    public void withdraw(double amount)
    {
       double newBalance = balance - amount;
       balance = newBalance;
```

Continued…

File: BankAccount.java

```java
   }

   /**
      Gets the current balance of the bank account.
      @return the current balance
   */
   public double getBalance()
   {
      return balance;
   }

   private int accountNumber;
   private double balance;
}
```

Output

```
size=3
first account number=1008
last account number=1729
```

Self Check

1. **How do you construct an array of 10 strings? An array list of strings?**
2. **What is the content of `names` after the following statements?**

```
ArrayList<String> names = new ArrayList<String>();
names.add("A");
names.add(0, "B");
names.add("C");
names.remove(1);
```

Answers

1.

```
new String[10];
new ArrayList<String>();
```

2. **names contains the strings "B" and "C" at positions 0 and 1**

Wrappers

- **You cannot insert primitive types directly into array lists**
- **To treat primitive type values as objects, you must use wrapper classes:**

```
ArrayList<Double> data = new ArrayList<Double>();
data.add(29.95);
double x = data.get(0);
```

Continued…

Wrappers

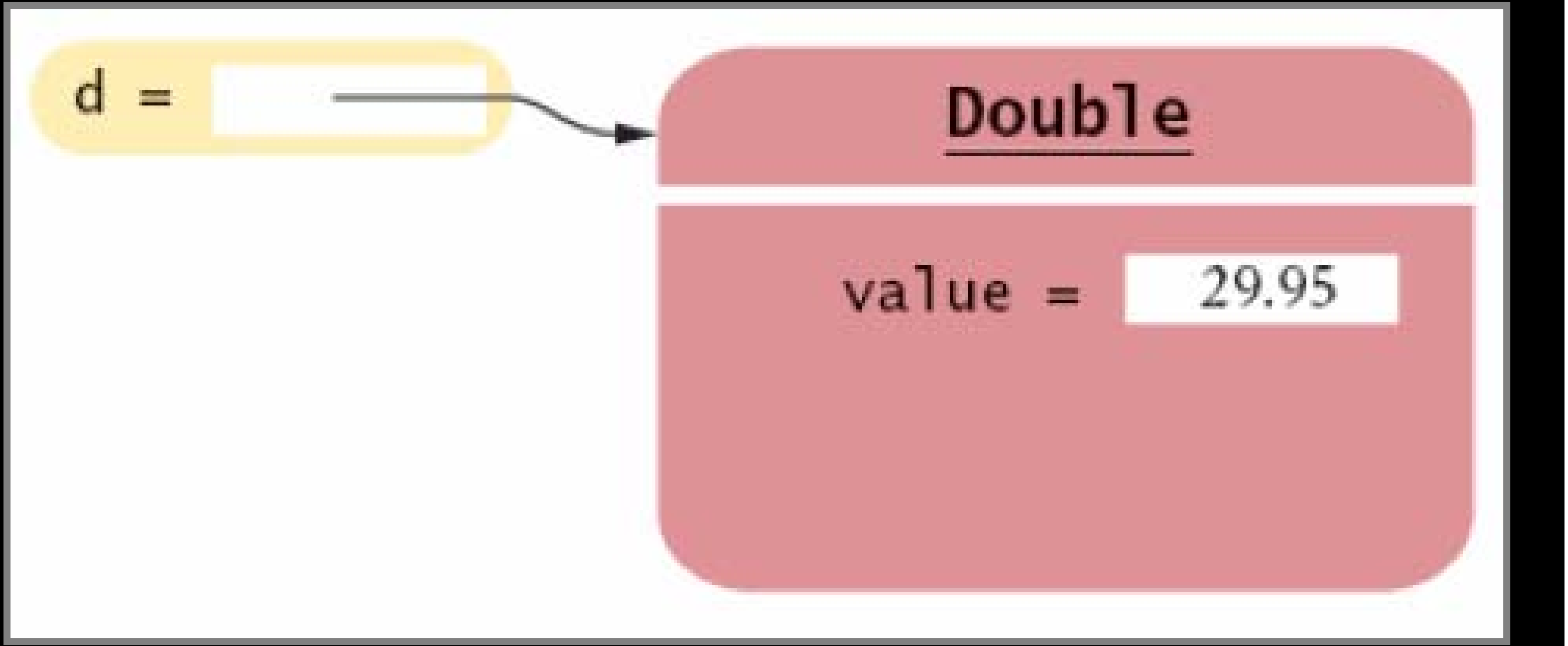
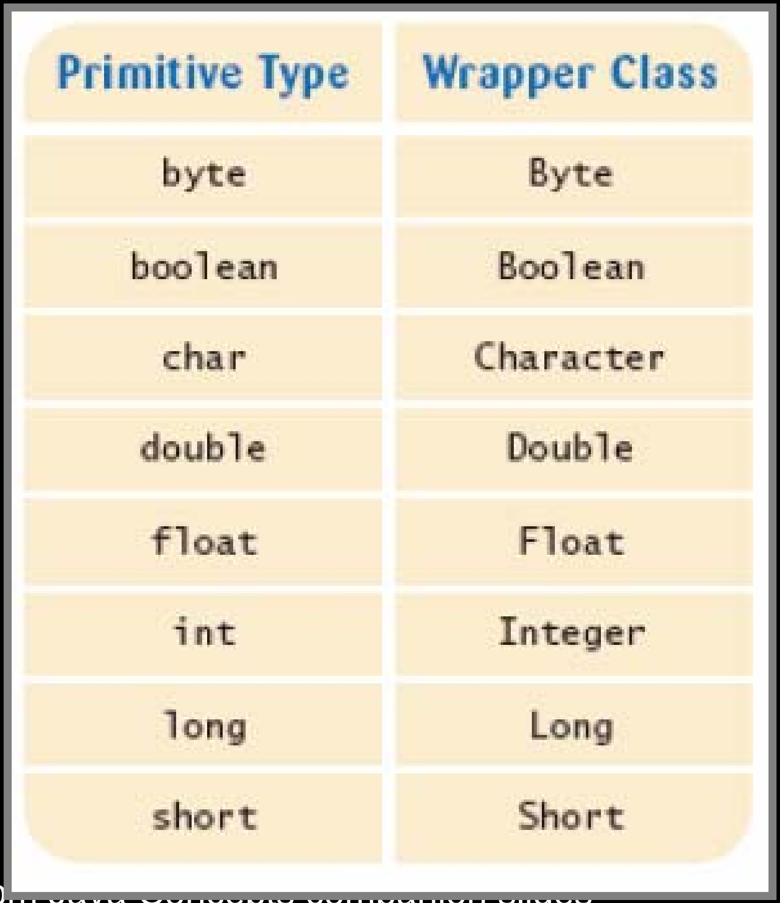


Figure 5:
An Object of a Wrapper Class

Wrappers

- There are wrapper classes for all eight primitive types



Primitive Type	Wrapper Class
byte	Byte
boolean	Boolean
char	Character
double	Double
float	Float
int	Integer
long	Long
short	Short

Auto-boxing

- **Auto-boxing: Starting with Java 5.0, conversion between primitive types and the corresponding wrapper classes is automatic.**

```
Double d = 29.95; // auto-boxing; same as Double d =
      new Double(29.95);
double x = d; // auto-unboxing; same as double x =
      d.doubleValue();
```

Continued…

Auto-boxing

- **Auto-boxing even works inside arithmetic expressions**

```
Double e = d + 1;
```

Means:

- auto-unbox `d` into a `double`
- add 1
- auto-box the result into a new `Double`
- store a reference to the newly created wrapper object in `e`

Self Check

1. What is the difference between the types `double` and `Double`?
2. Suppose data is an `ArrayList<Double>` of size > 0. How do you increment the element with index 0?

Answers

1. `double` is one of the eight primitive types. `Double` is a class type.
2. `data.set(0, data.get(0) + 1);`

The Generalized `for` Loop

- **Traverses all elements of a collection:**

```
double[] data = . . .;
double sum = 0;
for (double e : data) // You should read this loop as
      "for each e in data"
{
   sum = sum + e;
}
```

Continued…

The Generalized `for` Loop

- **Traditional alternative:**

```
double[] data = . . .;
double sum = 0;
for (int i = 0; i < data.length; i++)
{
   double e = data[i];
   sum = sum + e;
}
```

The Generalized `for` Loop

- Works for `ArrayLists` too:

```
ArrayList<BankAccount> accounts = . . . ;
double sum = 0;
for (BankAccount a : accounts)
{
   sum = sum + a.getBalance();
}
```

The Generalized `for` Loop

- Equivalent to the following ordinary `for` loop:

```
double sum = 0;
for (int i = 0; i < accounts.size(); i++)
{
   BankAccount a = accounts.get(i);
   sum = sum + a.getBalance();
}
```

Syntax 8.3: The "for each" Loop

```
 for (Type variable : collection)
    statement
```

Example:

```
 for (double e : data)
    sum = sum + e;
```

Purpose:

To execute a loop for each element in the collection. In each iteration, the variable is assigned the next element of the collection. Then the statement is executed.

Self Check

1. Write a "for each" loop that prints all elements in the array data
2. Why is the "for each" loop not an appropriate shortcut for the following ordinary `for` loop?

```
for (int i = 0; i < data.length; i++) data[i] = i * i;
```

Answers

1.
   ```
   for (double x : data) System.out.println(x);
   ```

2. The loop writes a value into `data[i]`. The "for each" loop does not have the index variable `i`.

Simple Array Algorithms: Counting Matches

- Check all elements and count the matches until you reach the end of the array list.

```
public class Bank
{
   public int count(double atLeast)
   {
      int matches = 0;
      for (BankAccount a : accounts)
      {
         if (a.getBalance() >= atLeast) matches++;
            // Found a match
      }
      return matches;
   }
   . . .
   private ArrayList<BankAccount> accounts;
}
```

Simple Array Algorithms: Finding a Value

- **Check all elements until you have found a match.**

```
public class Bank
{
   public BankAccount find(int accountNumber)
   {
      for (BankAccount a : accounts)
      {
         if (a.getAccountNumber() == accountNumber) // Found a match
            return a;
      }
      return null; // No match in the entire array list
   }
   . . .
}
```

Simple Array Algorithms: Finding the Maximum or Minimum

- Initialize a candidate with the starting element
- Compare candidate with remaining elements
- Update it if you find a larger or smaller value

Continued…

Simple Array Algorithms: Finding the Maximum or Minimum

- **Example:**

```
BankAccount largestYet = accounts.get(0);
for (int i = 1; i < accounts.size(); i++)
{
   BankAccount a = accounts.get(i);
   if (a.getBalance() > largestYet.getBalance())
      largestYet = a;
}
return largestYet;
```

Simple Array Algorithms: Finding the Maximum or Minimum

- Works only if there is at least one element in the array list
- If list is empty, return null

```
if (accounts.size() == 0) return null;
BankAccount largestYet = accounts.get(0);
. . .
```

File Bank.java

```java
import java.util.ArrayList;

/**
   This bank contains a collection of bank accounts.
*/
public class Bank
{
   /**
      Constructs a bank with no bank accounts.
   */
   public Bank()
   {
      accounts = new ArrayList<BankAccount>();
   }

   /**
      Adds an account to this bank.
      @param a the account to add
   */
```

Continued…

File Bank.java

```
   public void addAccount(BankAccount a)
   {
      accounts.add(a);
   }

   /**
      Gets the sum of the balances of all accounts in this bank.
      @return the sum of the balances
   */
   public double getTotalBalance()
   {
      double total = 0;
      for (BankAccount a : accounts)
      {
         total = total + a.getBalance();
      }
      return total;
   }

```

Continued…

File `Bank.java`

```
    /**
       Counts the number of bank accounts whose balance is at
       least a given value.
       @param atLeast the balance required to count an account
       @return the number of accounts having least the given
// balance
    */
    public int count(double atLeast)
    {
       int matches = 0;
       for (BankAccount a : accounts)
       {
          if (a.getBalance() >= atLeast) matches++; // Found
// a match
       }
       return matches;
    }

```

Continued…

File Bank.java

```java
/**
   Finds a bank account with a given number.
   @param accountNumber the number to find
   @return the account with the given number, or null
   if there is no such account
*/
public BankAccount find(int accountNumber)
{
   for (BankAccount a : accounts)
   {
      if (a.getAccountNumber() == accountNumber)
      // Found a match
         return a;
   }
   return null; // No match in the entire array list
}

```

Continued…

File Bank.java

```java
   /**
      Gets the bank account with the largest balance.
      @return the account with the largest balance, or
      null if the bank has no accounts
   */
   public BankAccount getMaximum()
   {
      if (accounts.size() == 0) return null;
      BankAccount largestYet = accounts.get(0);
      for (int i = 1; i < accounts.size(); i++)
      {
         BankAccount a = accounts.get(i);
         if (a.getBalance() > largestYet.getBalance())
            largestYet = a;
      }
      return largestYet;
   }

   private ArrayList<BankAccount> accounts;
}
```

File BankTester.java

```java
/**
   This program tests the Bank class.
*/
public class BankTester
{
   public static void main(String[] args)
   {
      Bank firstBankOfJava = new Bank();
      firstBankOfJava.addAccount(new BankAccount(1001, 20000));
      firstBankOfJava.addAccount(new BankAccount(1015, 10000));
      firstBankOfJava.addAccount(new BankAccount(1729, 15000));

      double threshold = 15000;
      int c = firstBankOfJava.count(threshold);
      System.out.println(c + " accounts with balance >= "
  + threshold);
```

Continued…

File `BankTester.java`

```

      int accountNumber = 1015;
      BankAccount a = firstBankOfJava.find(accountNumber);
      if (a == null)
         System.out.println("No account with number "
                + accountNumber);
      else
         System.out.println("Account with number "
                + accountNumber
                + " has balance " + a.getBalance());

      BankAccount max = firstBankOfJava.getMaximum();
      System.out.println("Account with number "
            + max.getAccountNumber()
            + " has the largest balance.");
   }
}
```

Continued…

File BankTester.java

Output

```
2 accounts with balance >= 15000.0
Account with number 1015 has balance 10000.0
Account with number 1001 has the largest balance.
```

Self Check

1. What does the `find` method do if there are two bank accounts with a matching account number?
2. Would it be possible to use a "for each" loop in the `getMaximum` method?

Answers

1. It returns the first match that it finds
2. Yes, but the first comparison would always fail

Two-Dimensional Arrays

- **When constructing a two-dimensional array, you specify how many rows and columns you need:**

```
final int ROWS = 3;
final int COLUMNS = 3;
String[][] board = new String[ROWS][COLUMNS];
```

- **You access elements with an index pair** **`a[i][j]`**

```
board[i][j] = "x";
```

A Tic-Tac-Toe Board

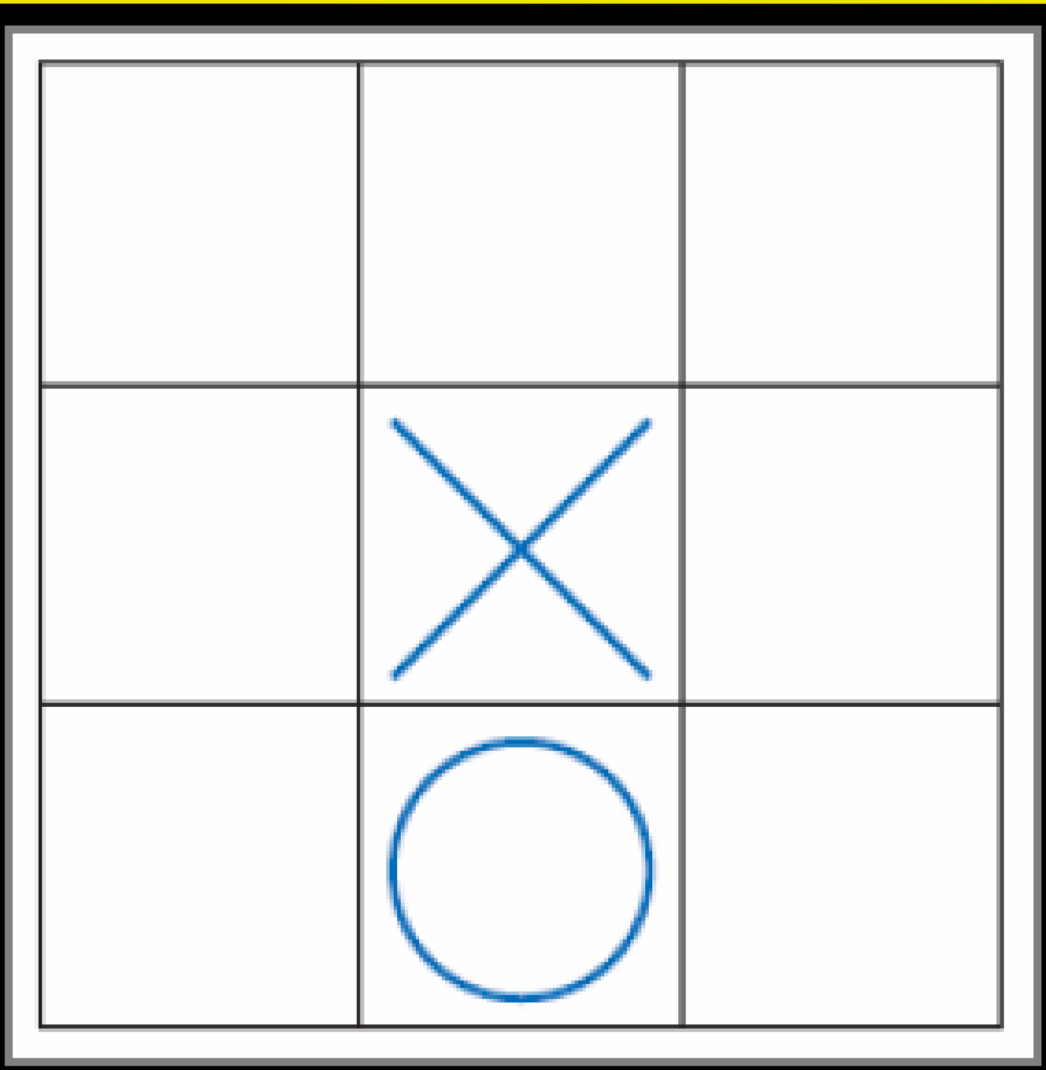


Figure 6:
A Tic-Tac-Toe Board

Traversing Two-Dimensional Arrays

- It is common to use two nested loops when filling or searching:

```
for (int i = 0; i < ROWS; i++)
   for (int j = 0; j < COLUMNS; j++)
      board[i][j] = " ";
```

File TicTacToe.java

```java
/**
   A 3 x 3 tic-tac-toe board.
*/
public class TicTacToe
{
   /**
      Constructs an empty board.
   */
   public TicTacToe()
   {
      board = new String[ROWS][COLUMNS];
      // Fill with spaces
      for (int i = 0; i < ROWS; i++)
         for (int j = 0; j < COLUMNS; j++)
            board[i][j] = " ";
   }

```

Continued…

File TicTacToe.java

```java
   /**
      Sets a field in the board. The field must be unoccupied.
      @param i the row index
      @param j the column index
      @param player the player ("x" or "o")
   */
   public void set(int i, int j, String player)
   {
      if (board[i][j].equals(" "))
         board[i][j] = player;
   }

   /**
      Creates a string representation of the board, such as
      |x  o|
      |  x |
      |   o|
      @return the string representation
   */
```

Continued…

File `TicTacToe.java`

```java
   public String toString()
   {
      String r = "";
      for (int i = 0; i < ROWS; i++)
      {
         r = r + "|";
         for (int j = 0; j < COLUMNS; j++)
            r = r + board[i][j];
         r = r + "|\n";
      }
      return r;
   }

   private String[][] board;
   private static final int ROWS = 3;
   private static final int COLUMNS = 3;
}
```

File TicTacToeTester.java

```java
import java.util.Scanner;

/**
   This program tests the TicTacToe class by prompting the
   user to set positions on the board and printing out the
   result.
*/
public class TicTacToeTester
{
   public static void main(String[] args)
   {
      Scanner in = new Scanner(System.in);
      String player = "x";
      TicTacToe game = new TicTacToe();
      boolean done = false;
      while (!done)
      {
```

Continued…

File TicTacToeTester.java

```java
         System.out.print(game.toString());
         System.out.print(
               "Row for " + player + " (-1 to exit): ");
         int row = in.nextInt();
         if (row < 0) done = true;
         else
         {
            System.out.print("Column for " + player + ": ");
            int column = in.nextInt();
            game.set(row, column, player);
            if (player.equals("x"))
               player = "o";
            else
               player = "x";
         }
      }
   }
}
```

Continued…

Output

```
| |
| |
| |
Row for x (-1 to exit): 1

Column for x: 2

|  |

| x|

|

Row for o (-1 to exit): 0

Column for o: 0

|o |

| x|

|  |

Row for x (-1 to exit): -1
```

Self Check

1. How do you declare and initialize a 4-by-4 array of integers?
2. How do you count the number of spaces in the tic-tac-toe board?

Answers

1.

```
int[][] array = new int[4][4];
```

2.

```
int count = 0;
for (int i = 0; i < ROWS; i++)
   for (int j = 0; j < COLUMNS; j++)
      if (board[i][j] == ' ') count++;
```

Copying Arrays: Copying Array References

- **Copying an array variable yields a second reference to the same array**

```
double[] data = new double[10];
// fill array . . .
double[] prices = data;
```

Continued…

Copying Arrays: Copying Array References

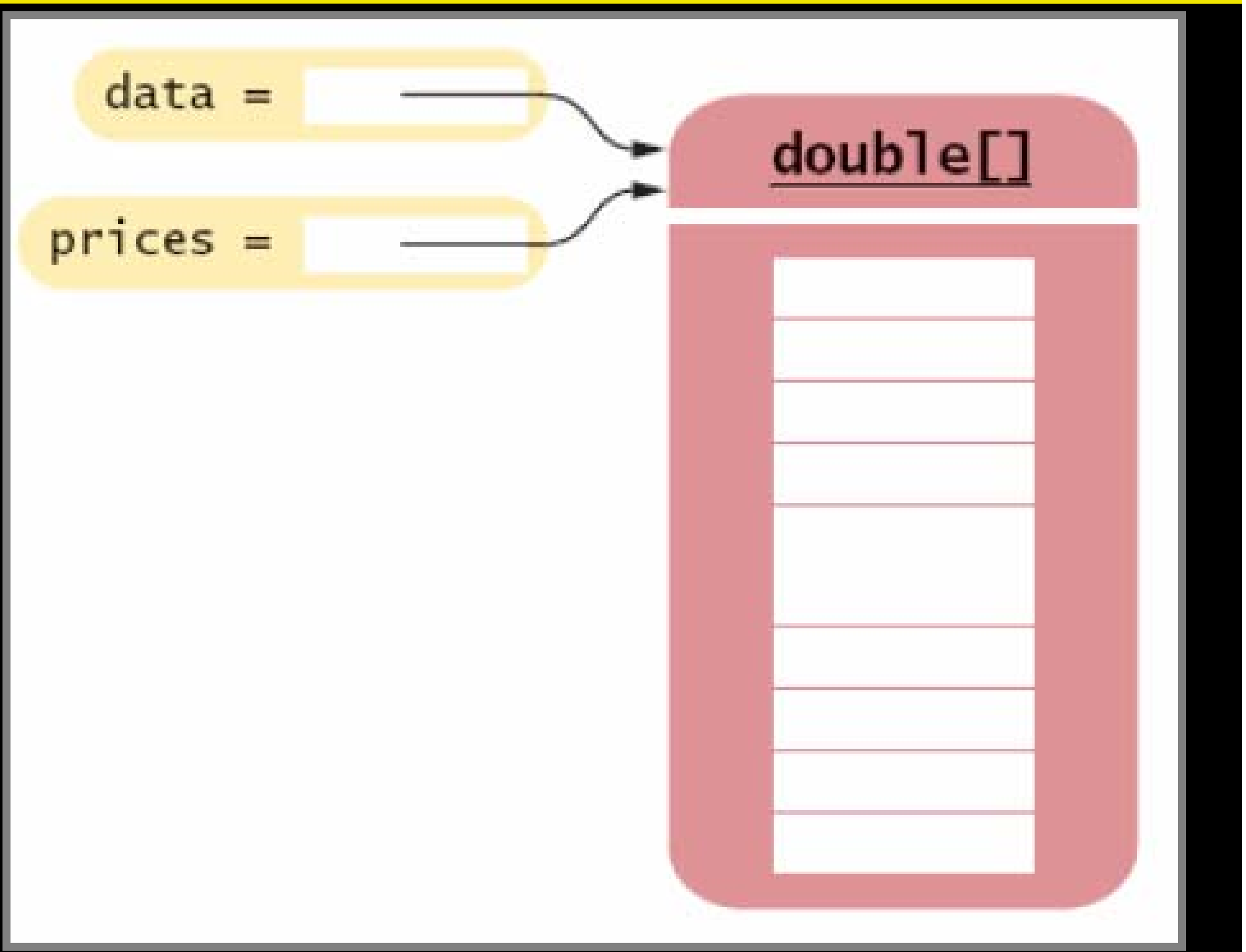


Figure 7: Two References to the Same Array

Copying Arrays: Cloning Arrays

- **Use `clone` to make true copy**

```
double[] prices = (double[]) data.clone();
```

Continued…

Copying Arrays: Cloning Arrays

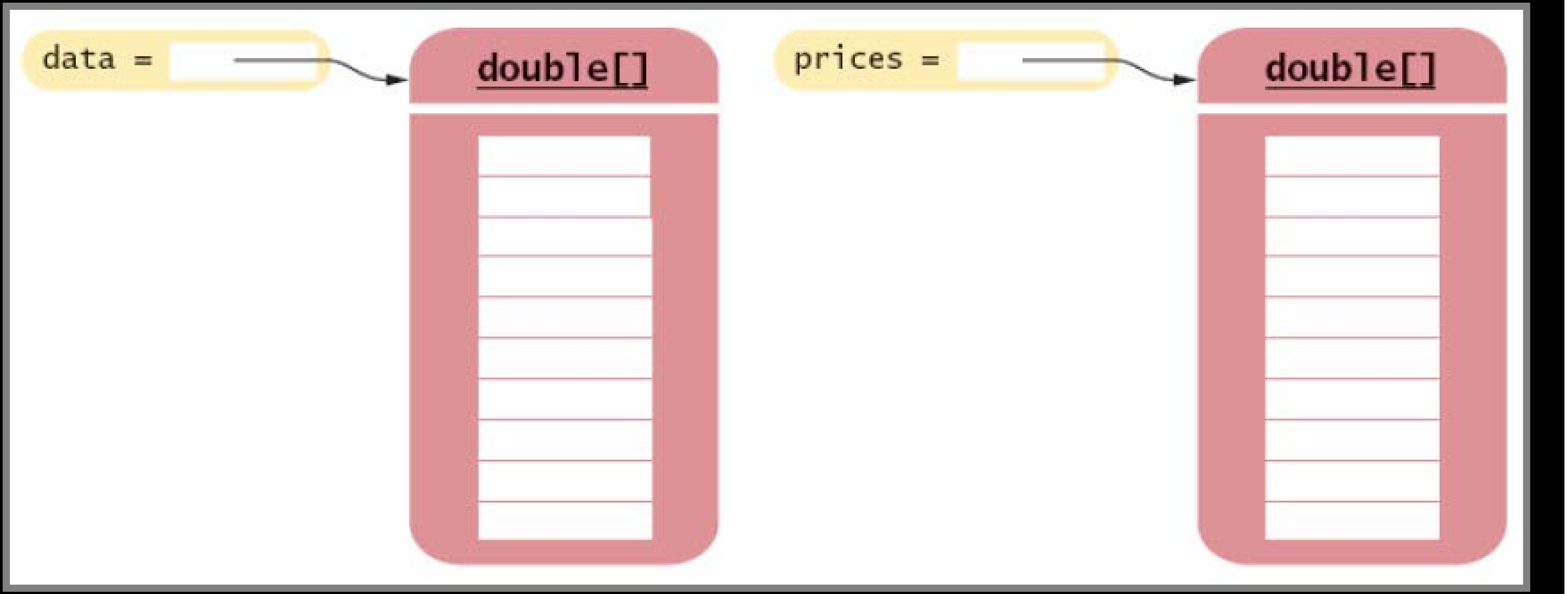


Figure 8: Cloning an Array

Copying Arrays: Copying Array Elements

```
System.arraycopy(from, fromStart, to, toStart, count);
```

Continued…

Copying Arrays: Copying Array Elements

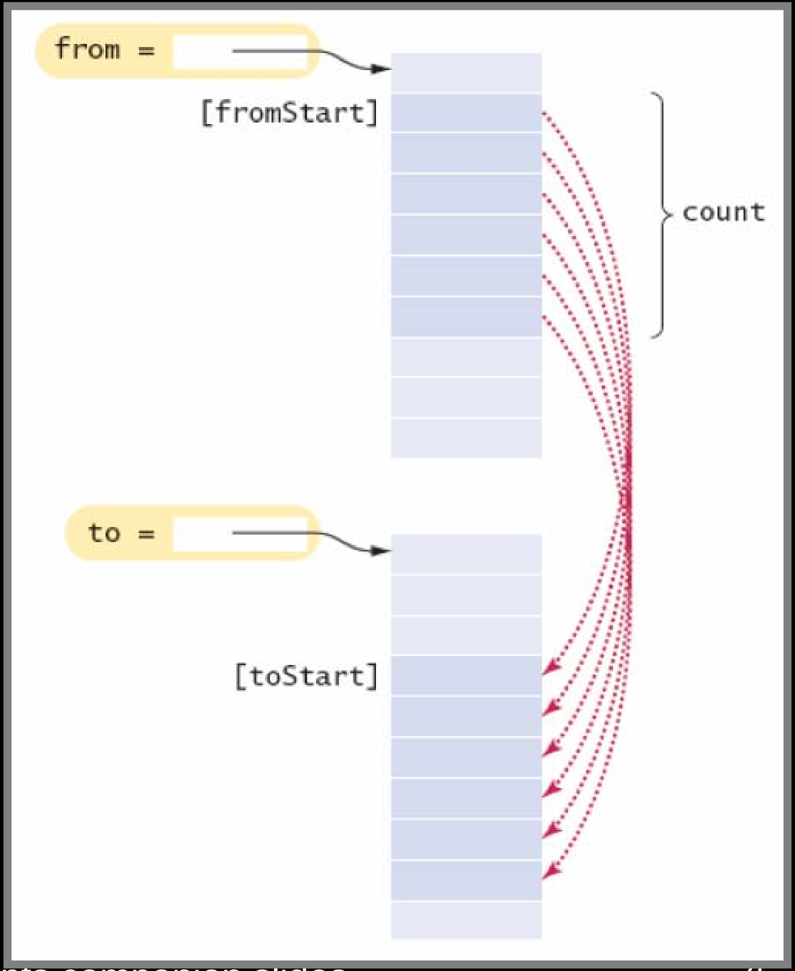


Figure 9:
The System.`arraycopy` Method

Adding an Element to an Array

```
System.arraycopy(data, i, data, i + 1, data.length - i - 1);
data[i] = x;
```

Continued…

Adding an Element to an Array

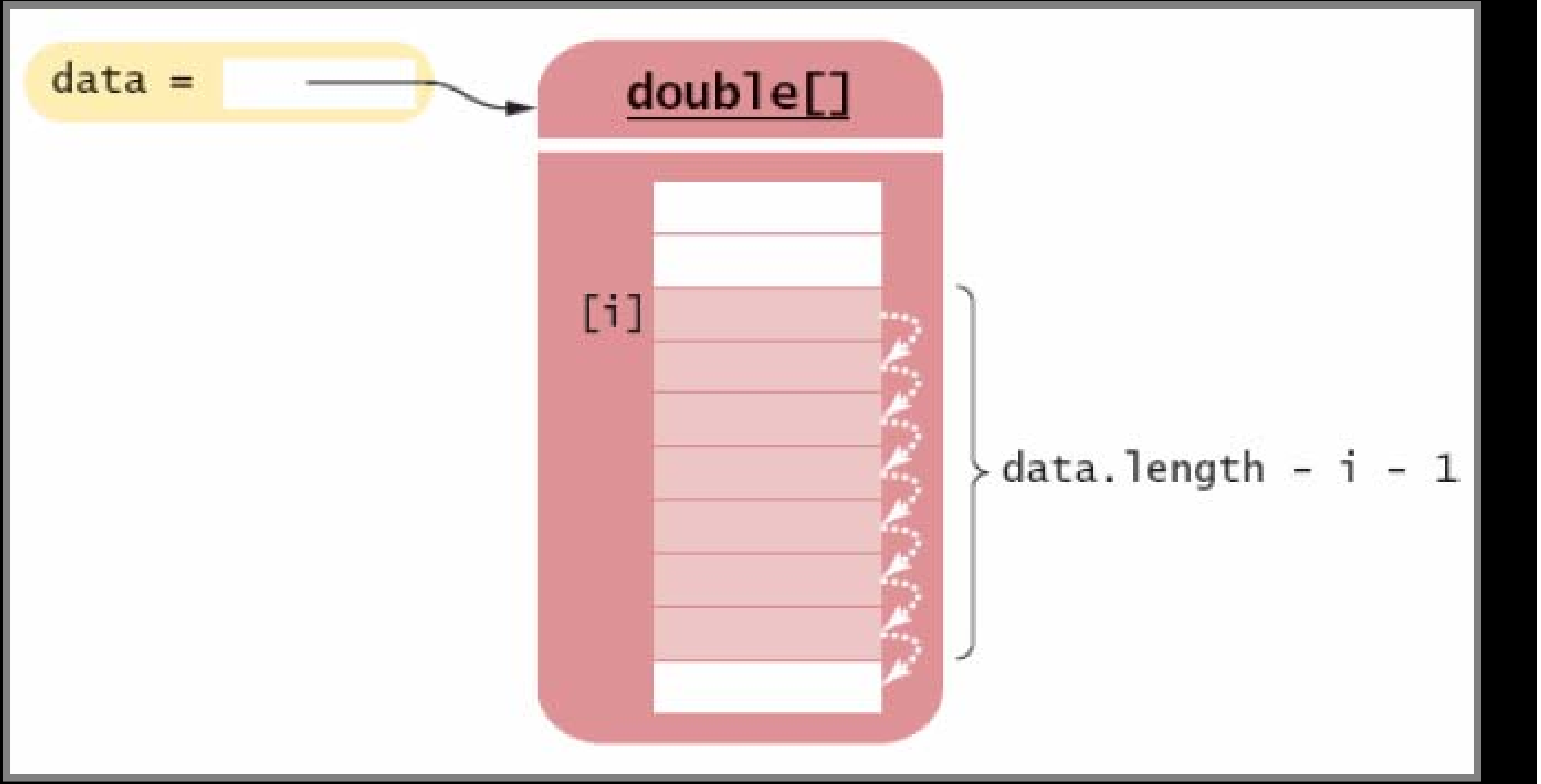


Figure 10:
Inserting a New Element Into an Array

Removing an Element from an Array

```
System.arraycopy(data, i + 1, data, i, data.length - i - 1);
```

Continued…

Removing an Element from an Array

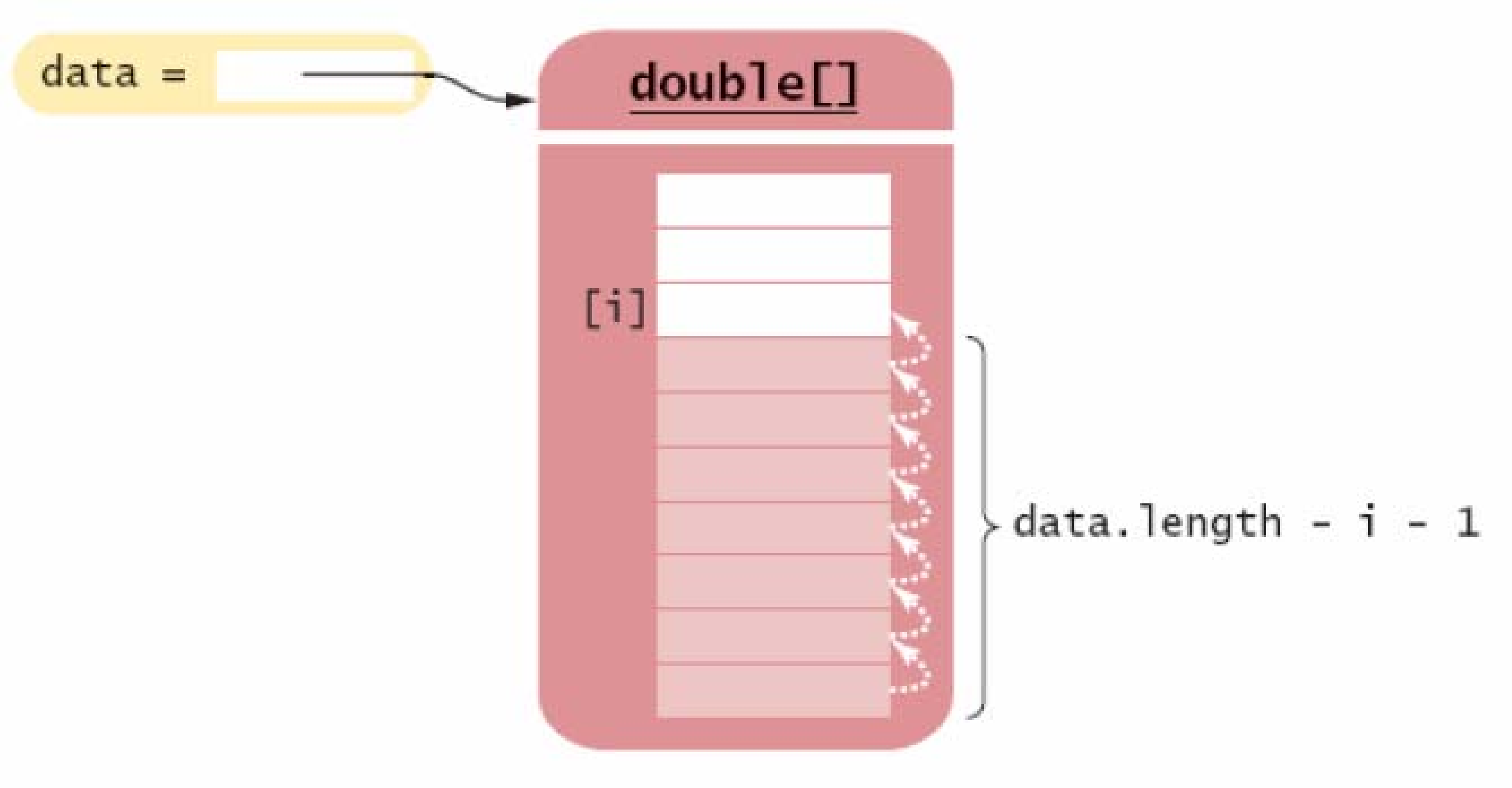


Figure 11
Removing an Element from an Array

Growing an Array

- **If the array is full and you need more space, you can grow the array:**

1. Create a new, larger array.

```
double[] newData = new double[2 * data.length];
```

2. Copy all elements into the new array

```
System.arraycopy(data, 0, newData, 0, data.length);
```

3. Store the reference to the new array in the array variable

```
data = newData;
```

Growing an Array

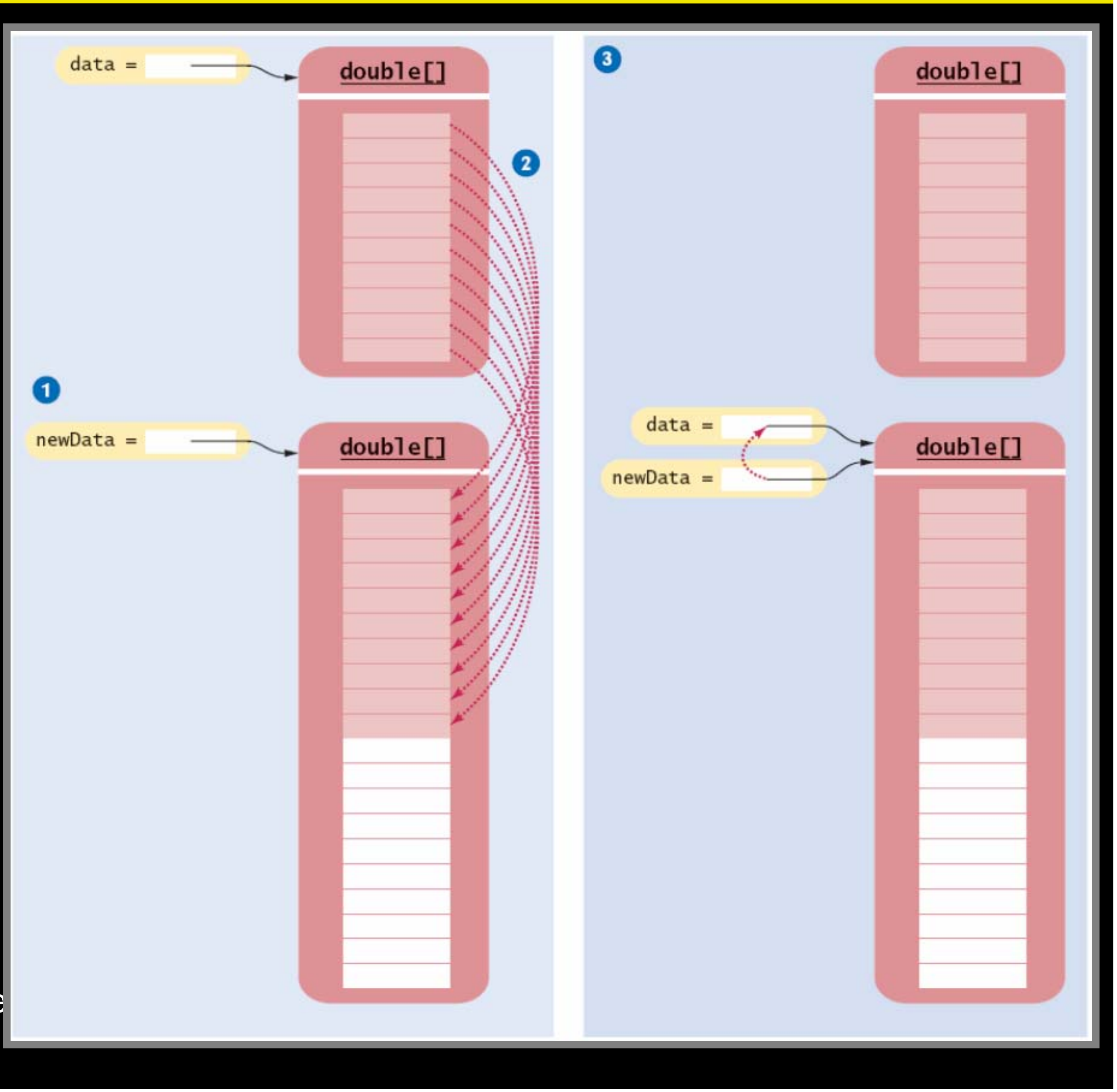


Figure 12:
Growing an Array

Self Check

1. How do you add or remove elements in the middle of an array list?
2. Why do we double the length of the array when it has run out of space rather than increasing it by one element?

Answers

1. Use the `insert` and `remove` methods.
2. Allocating a new array and copying the elements is time-consuming. You wouldn't want to go through the process every time you add an element.

Make Parallel Arrays into Arrays of Objects

-
```
// Don't do this
int[] accountNumbers;
double[] balances;
```

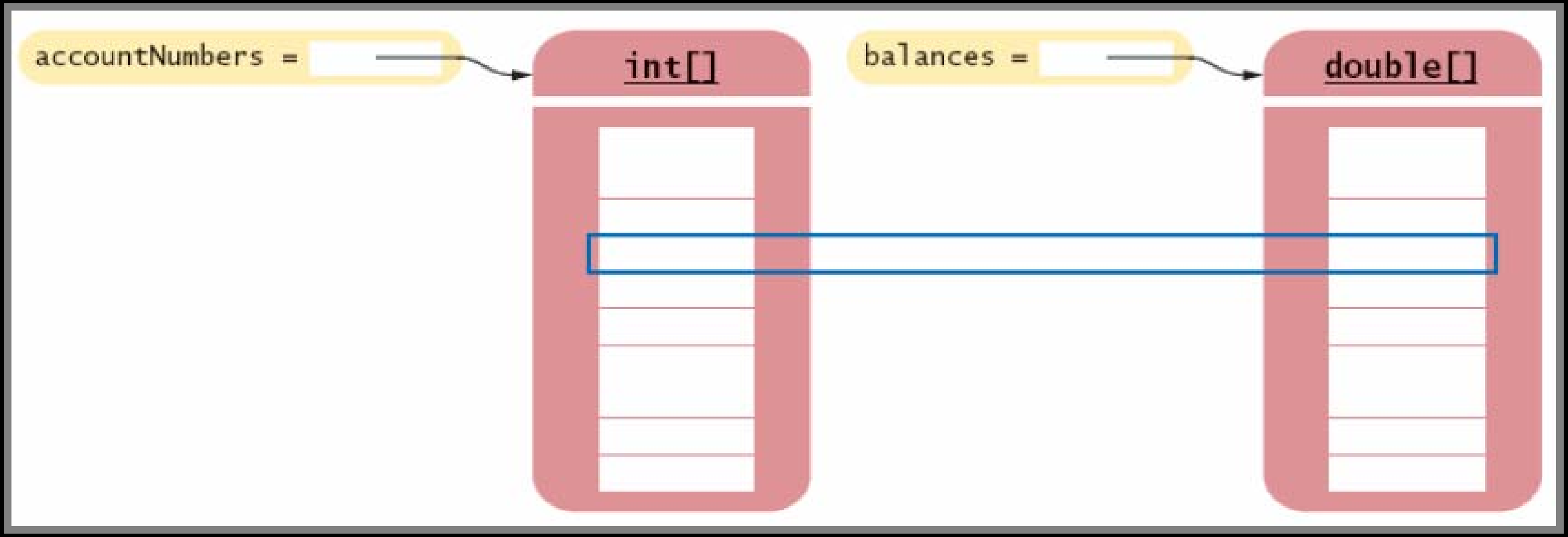


Figure 13:
Avoid Parallel Arrays

Make Parallel Arrays into Arrays of Objects

- **Avoid parallel arrays by changing them into arrays of objects:**

```
BankAccount[] = accounts;
```

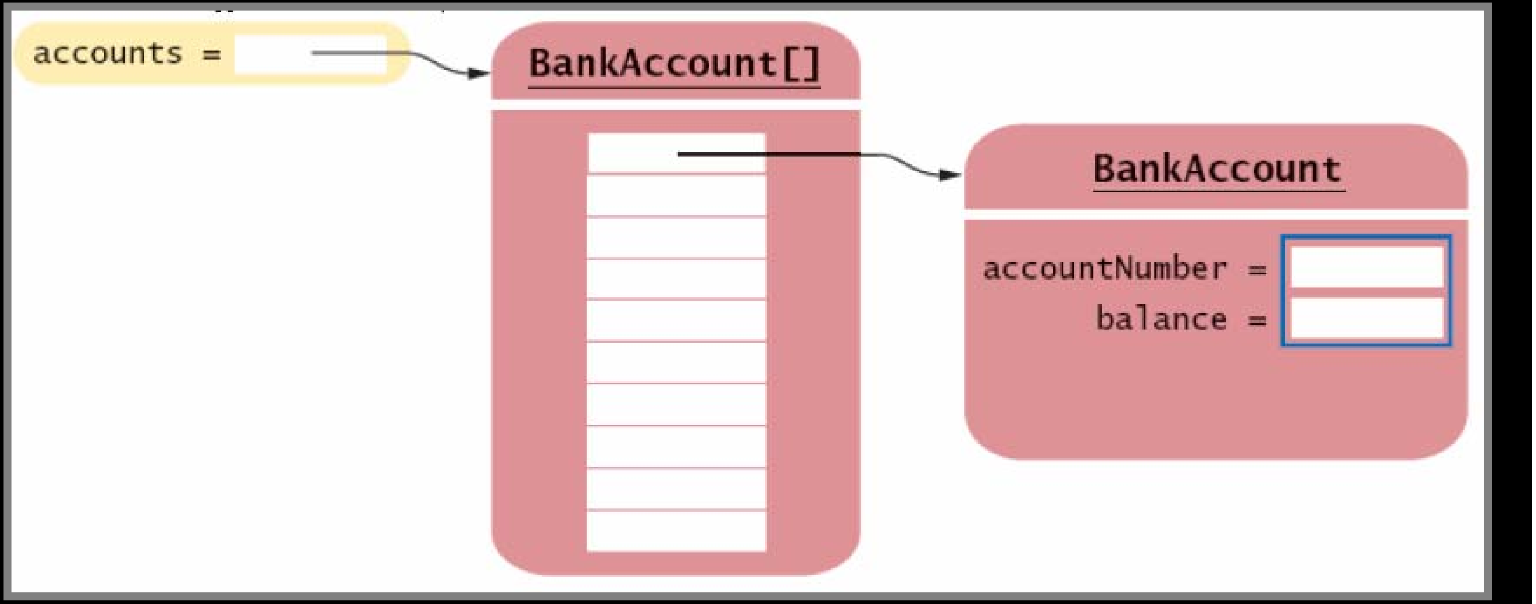


Figure 14:
Reorganizing Parallel Arrays into Arrays of Objects

Partially Filled Arrays

- Array length = maximum number of elements in array
- Usually, array is partially filled
- Need companion variable to keep track of current size
- Uniform naming convention:

```
final int DATA_LENGTH = 100;
double[] data = new double[DATA_LENGTH];
int dataSize = 0;
```

Continued…

Partially Filled Arrays

- **Update `dataSize` as array is filled:**

```
data[dataSize] = x;
dataSize++;
```

Partially Filled Arrays

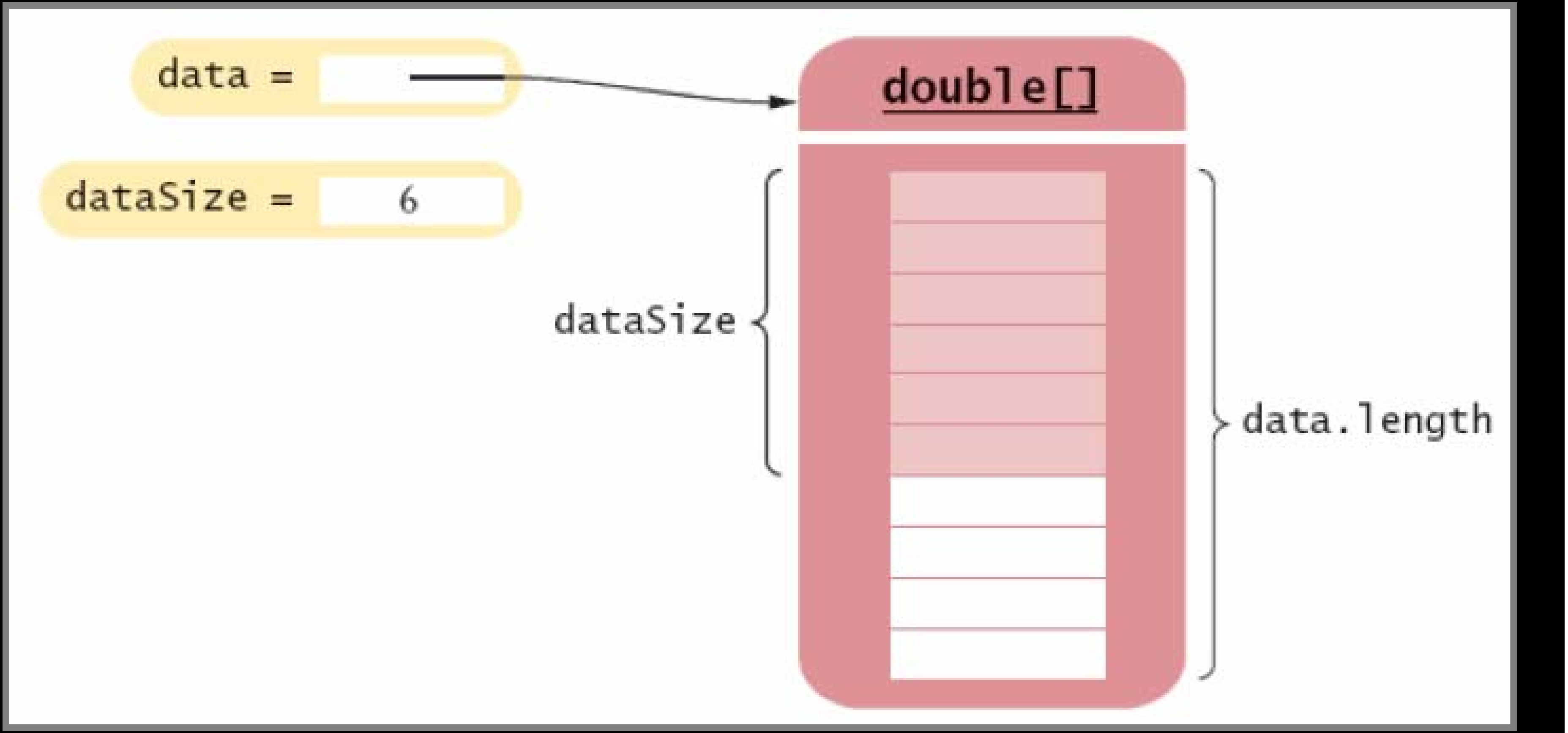


Figure 15:
A Partially Filled Array

An Early Internet Worm

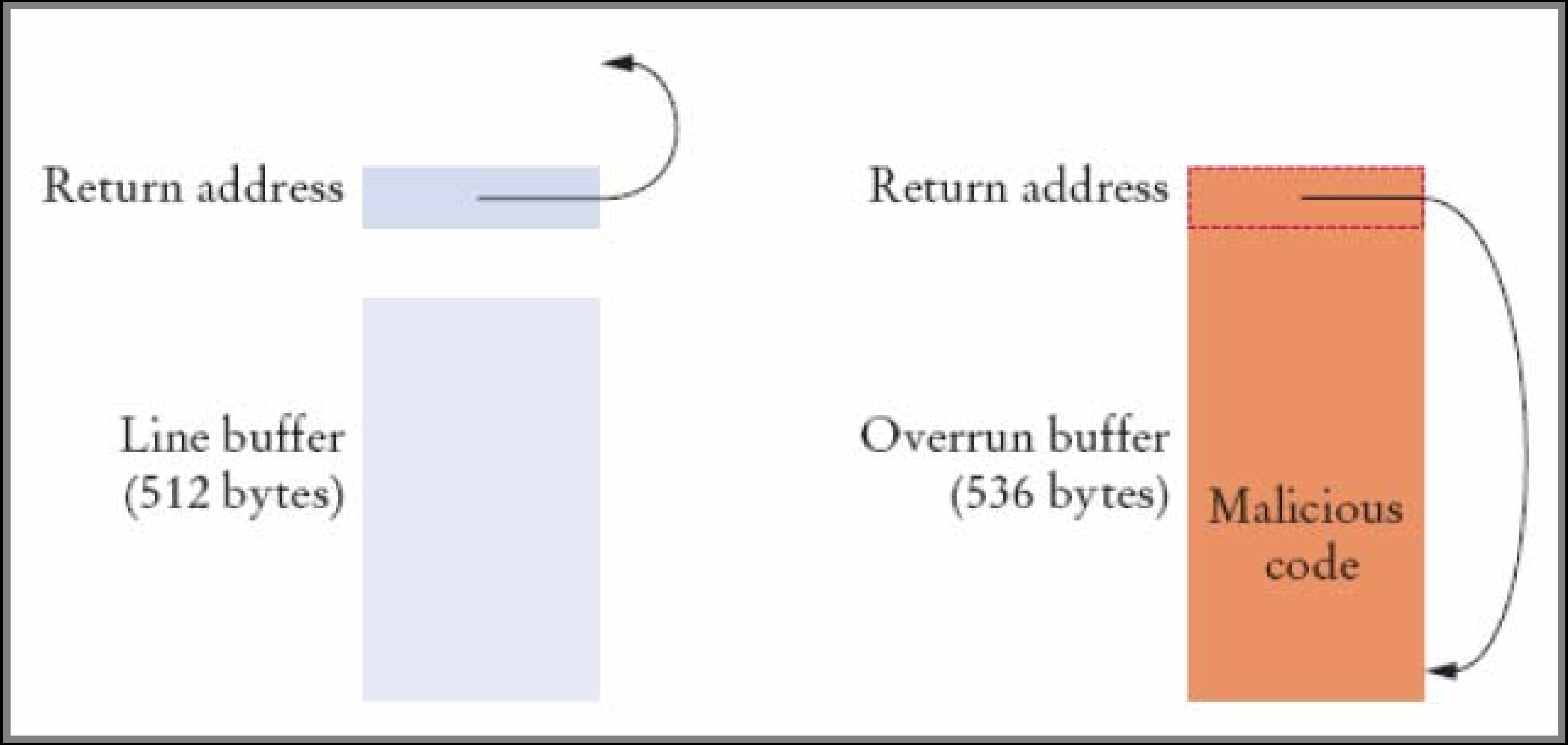


Figure 16:
A "Buffer Overrun" Attack