Course Material Usage Rules

• PowerPoint slides for use only in for-credit courses at degree-granting institutions
  – Slides not permitted for commercial training courses except when taught by coreservlets.com (see http://courses.coreservlets.com).

• Slides may be modified by instructor
  – But please retain reference to coreservlets.com

• Instructor may give PDF or hardcopy to students
  – But should protect the PowerPoint files
Java 5 – What’s New: Syntax and Utilities II

Originals of Slides and Source Code for Examples:
http://courses.coreservlets.com/Course-Materials/java5.html

Customized Java EE Training: http://courses.coreservlets.com/
Servlets, JSP, JSF 2.0, Struts, Ajax, GWT 2.0, Spring, Hibernate, SOAP & RESTful Web Services, Java 6.
Developed and taught by well-known author and developer. At public venues or onsite at your location.
For live Java EE training, please see training courses at http://courses.coreservlets.com/.

Servlets, JSP, Struts, JSF 1.x, JSF 2.0, Ajax (with jQuery, Dojo, Prototype, Ext-JS, Google Closure, etc.), GWT 2.0 (with GXT), Java 5, Java 6, SOAP-based and RESTful Web Services, Spring, Hibernate/JPA, and customized combinations of topics.

Taught by the author of Core Servlets and JSP, More Servlets and JSP, and this tutorial. Available at public venues, or customized versions can be held on-site at your organization. Contact hall@coreservlets.com for details.
Topics in This Section

• Data structures
  – ArrayList
  – LinkedList
  – HashMap
• Generics
• printf
• varargs
• String vs. StringBuilder
Lists and Generics
Problem

- You want to make an ordered list of objects. But, even after you get the first few elements, you don’t know how many more you will have.
  - Thus, you can’t use an array, since the size of arrays must be known at the time that you allocate it. (Although Java arrays are better than C++ arrays since the size does not need to be a compile-time constant)

Solution

- Use ArrayList or LinkedList: they stretch as you add elements to them

Notes

- The two options give the same results for the same operations, but differ in performance
Syntax: ArrayList & LinkedList

- **Summary of operations**
  - Create empty list
    - `new ArrayList<Type`() or `new LinkedList<Type>()`
      - Note that you need "import java.util.*;" at the top of file
  - Add entry to end
    - `add(value)` (adds to end) or `add(index, value)`
  - Retrieve n-th element
    - `get(index)`
  - Check if element exists in list
    - `contains(element)`
  - Remove element
    - `remove(index)` or `remove(element)`
  - Find the number of elements
    - `size()`
import java.util.*; // Don't forget this import

public class ListTest2 {
    public static void main(String[] args) {
        List<String> entries = new ArrayList<String>();
        double d;
        while((d = Math.random()) > 0.1) {
            entries.add("Value: " + d);
        }
        for(String entry: entries) {
            System.out.println(entry);
        }
    }
}
ArrayList Example: Output

> java ListTest2
Value: 0.6374760850618444
Value: 0.9159907384916878
Value: 0.8093728146584014
Value: 0.7177611068808302
Value: 0.9751541794430284
Value: 0.2655587762679209
Value: 0.3135791999033012
Value: 0.44624152771013836
Value: 0.7585420756498766
## Comparing ArrayList and LinkedList Performance

<table>
<thead>
<tr>
<th></th>
<th>Array with Copying (ArrayList)</th>
<th>List of Pointers (LinkedList)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Insert at beginning</td>
<td>O(N)</td>
<td>O(1)</td>
</tr>
<tr>
<td>Insert at end</td>
<td>O(1) if space O(N) if not O(1) amortized time</td>
<td>O(1)</td>
</tr>
<tr>
<td>Access Nth Element</td>
<td>O(1)</td>
<td>O(N)</td>
</tr>
</tbody>
</table>
Using Generics

- General steps
  - Find a data structure that accepts Object(s)
    - ArrayList, LinkedList, HashMap, HashSet, Stack
  - Declare the data structure with the type(s) in angle brackets immediately after class name
    - List<String> names = new ArrayList<String>();
    - Map<String,Person> employees = new HashMap<String,Person>();
  - Insert objects of the appropriate type
    - names.add("Some String");
    - employees.put(person.getEmployeeId(), person);
  - No typecast required on removal
    - String firstName = names.get(0);
    - Person p1 = employees.get("a1234");
import java.util.*;

public class ListTest1 {
    public static void main(String[] args) {
        List entries = new ArrayList();
        double d;
        while((d = Math.random()) > 0.1) {
            entries.add("Value: " + d);
        }
        String entry;
        for(int i=0; i<entries.size(); i++) {
            entry = (String)entries.get(i);
            System.out.println(entry);
        }
    }
}

import java.util.*;

public class ListTest2 {
    public static void main(String[] args) {
        List<String> entries = new ArrayList<String>();
        double d;
        while((d = Math.random()) > 0.1) {
            entries.add("Value: " + d);
        }
        for(String entry: entries) {
            System.out.println(entry);
        }
    }
}
**Autoboxing**

- **You cannot insert primitives into tables, lists, or anything else expecting an Object**
  - Java provides wrapper types for this purpose (int → Integer, etc.)
- **In Java 5+, system converts automatically**
  - Performance Warning
    - Every insert converts to wrapper type (Integer above)
    - Every retrieval converts to primitive type (int above)
    - Use arrays for performance-critical access

<table>
<thead>
<tr>
<th>Old</th>
<th>New</th>
</tr>
</thead>
</table>
| List nums = new ArrayList();  
  int i = someCalculation();  
  nums.add(new Integer(i));  
  ...  
  Integer val =  
    (Integer)nums.get(someIndex);  
  int num = val.intVal() + 1;  
  nums.add(new Integer(num)); | List<Integer> nums =  
  new ArrayList<Integer>();  
  nums.add(someCalculation());  
  ...  
  int num =  
    nums.get(someIndex);  
  nums.add(num + 1); |
Maps
(Also called “Lookup Tables” or “Associative Arrays”)
HashMap

• **HashMap provides simple lookup table**
  - Use “put” to store data
    
    ```java
    Map<String, Person> employees = new HashMap<String, Person>();
    Person p1 = new Person("a1234", "Larry", "Ellison");
    employees.put(p1.getEmployeeId(), p1);
    ```
  - Use “get” to retrieve data
    
    ```java
    Person p = employees.get("a1234");
    ```
    - Returns null if no match

• **Performance**
  - Insert and retrieval time are independent of the number of entries in the table, i.e., O(1). (How do they do that?)
    • But Java has other Map types with different performance characteristics and features
public class StateMap {
    private Map<String, String> stateMap;

    public StateMap() {
        stateMap = new HashMap<String, String>();
        for (String[] state : stateArray) {
            stateMap.put(state[0], state[1]);
        }
    }

    public Map<String, String> getStateMap() {
        return stateMap;
    }

    public String[][] getStateArray() {
        return stateArray;
    }

    private String[][] stateArray = {
        { "Alabama", "AL" },
        { "Alaska", "AK" },
        { "Arizona", "AZ" }, ...
    }
}
public class MapTest {
    public static void main(String[] args) {
        StateMap states = new StateMap();
        Map<String, String> stateAbbreviationTable =
            states.getStateMap();
        Scanner inputScanner = new Scanner(System.in);
        System.out.println("Enter state names. " +
            "Hit RETURN to quit");
        String stateName;
        String abbreviation;
    }
}
HashMap Example: Finding State Abbreviations Based on State Names

```java
while(true) {
    System.out.print("State: ");
    stateName = inputScanner.nextLine();
    if (stateName.equals("")) {
        System.out.println("Come again soon.");
        break;
    }
    abbreviation =
        stateAbbreviationTable.get(stateName);
    if (abbreviation == null) {
        abbreviation = "Unknown";
    }
    System.out.println(abbreviation);
}
```
printf

Customized Java EE Training: http://courses.coreservlets.com/
Servlets, JSP, JSF 2.0, Struts, Ajax, GWT 2.0, Spring, Hibernate, SOAP & RESTful Web Services, Java 6. Developed and taught by well-known author and developer. At public venues or onsite at your location.
Formated Output: printf

- Takes a variable number of arguments
  - `System.out.printf("Formatting String", arg1, arg2, ...);`

- Advantages
  - Lets you insert values into output without much clumsier String concatenation.
  - Lets you control the width of results so things line up
  - Lets you control the number of digits after the decimal point in numbers, for consistent-looking output

- Very similar to C/C++ printf function
  - If you know printf in C/C++, you can probably use Java's printf immediately without reading any documentation
    - Although some additions in time formatting and locales
  - Use `String.format` to get the equivalent of C's `sprintf`
Simple Example: printf vs. println

• General idea
  – Each %s entry in formatting string is replaced by next argument in argument list. %n means newline.

• Example
  public static void printSomeStrings() {
    String firstName = "John";
    String lastName = "Doe";
    int numPets = 7;
    String petType = "chickens";
    System.out.printf("%s %s has %s %s.%n",
                       firstName, lastName, numPets, petType);
    System.out.println(firstName + " " + lastName + " has " + numPets + " " +
                        petType + ".");
  }

• Result:
  John Doe has 7 chickens.
  John Doe has 7 chickens.
Controlling Formatting

• **Different flags**
  - %s for strings, %f for floats/doubles, %t for dates, etc.
    * Unlike in C/C++, you can use %s for any type (even nums)

• **Various extra entries can be inserted**
  - To control width, number of digits, commas, justification, type of date format, and more

• **Complete details**
  - printf uses mini-language
    * Complete coverage would take an entire lecture
    * However, basic usage is straightforward
  - For complete coverage, see http://java.sun.com/j2se/1.6.0/docs/api/java/util/Formatter.html#syntax

• **Most common errors**
  - Using + instead of , between arguments (printf uses varargs)
  - Forgetting to add %n at the end if you want a newline (not automatic)
## Printf Formatting Options

<table>
<thead>
<tr>
<th>Stands For</th>
<th>Options</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>%s</strong></td>
<td>String. Can output any data type. If arg is Object, toString is called.</td>
<td>%widths&lt;br&gt;Gives min num of chars. Spaces added to left if needed.</td>
</tr>
<tr>
<td><strong>%d</strong></td>
<td>Decimal. Outputs whole number in base 10. Also %x and %o for hex and octal.</td>
<td>%widthd %,widthd&lt;br&gt;Gives min width; inserts commas.</td>
</tr>
<tr>
<td><strong>%f</strong></td>
<td>Floating point. Lets you line up decimal point and control precision.</td>
<td>%width.precisionf&lt;br&gt;%,width.precisionf&lt;br&gt;width includes comma and decimal point.</td>
</tr>
<tr>
<td><strong>%t</strong></td>
<td>Time (or date). %tA for day, %tB for month, %tY for year, and many more.</td>
<td>Date now = new Date();&lt;br&gt; printf(&quot;%tA, %tB, %tY&quot;, now, now, now)&lt;br&gt; outputs &quot;Thursday, November 17, 2005&quot;</td>
</tr>
<tr>
<td><strong>%n</strong></td>
<td>Outputs OS-specific end of line (linefeed on Linux, CR/LF pair on Windows)</td>
<td></td>
</tr>
</tbody>
</table>
Printf Example: Controlling Width and Precision

```java
public static void printSomeSalaries() {
    CEO[] softwareCEOs =
    { new CEO("Steve Jobs", 3.1234),
        new CEO("Scott McNealy", 45.5678),
        new CEO("Jeff Bezos", 567.982323),
        new CEO("Larry Ellison", 6789.0),
        new CEO("Bill Gates", 78901234567890.12)};
    System.out.println("SALARIES:");
    for(CEO ceo: softwareCEOs) {
        System.out.printf("%15s: $%,8.2f%n", 
            ceo.getName(), ceo.getSalary());
    }
}
```

SALARIES:

- Steve Jobs: $ 3.12
- Scott McNealy: $ 45.57
- Jeff Bezos: $ 567.98
- Larry Ellison: $6,789.00
- Bill Gates: $78,901,234,567,890.12
Printf Example: Controlling Width and Precision

public class CEO {
    private String name;
    private double salary; // In billions

    public CEO(String name, double salary) {
        this.name = name;
        this.salary = salary;
    }

    public String getName() { return(name); }

    public double getSalary() { return(salary); }
}
Varargs
Variable-Length Arguments

• The printf method takes any number of arguments
  – You could use overloading to define a few versions of printf with different argument lengths, but it takes any number of arguments

• To do this yourself, use "type ... variable"
  – variable becomes an array of given type
  – Only legal for final argument of method
  – Examples
    • public void printf(String format, Object ... arguments)
    • public int max(int ... numbers)
      – Can call max(1, 2, 3, 4, 5, 6) or max(someArrayOflnts)

• Use sparingly
  • You usually know how many arguments are possible
public class MathUtils {
    public static int min(int ... numbers) {
        int minimum = Integer.MAX_VALUE;
        for(int number: numbers) {
            if (number < minimum) {
                minimum = number;
            }
        }
        return(minimum);
    }

    public static void main(String[] args) {
        System.out.printf("Min of 2 nums: %d.%n", min(2,1));
        System.out.printf("Min of 7 nums: %d.%n", min(2,4,6,8,1,2,3));
    }
}
StringBuilder

Customized Java EE Training: http://courses.coreservlets.com/
Servlets, JSP, JSF 2.0, Struts, Ajax, GWT 2.0, Spring, Hibernate, SOAP & RESTful Web Services, Java 6.
Developed and taught by well-known author and developer. At public venues or onsite at your location.
String vs. StringBuilder

- **Strings are immutable (unmodifiable)**
  - Thus what appears to be String concatenation really involves copying the string on the left (oldString below)
    - String newString = oldString + "some extra stuff"
  - Never do String concatenation inside a loop that could be very long (i.e., more than about 100)

- **StringBuilder is mutable**
  - Build a StringBuilder from a String with new StringBuilder(someString)
  - Call append to append data to the end
  - Call toString to turn back into a string
  - Other methods: insert, replace, substring, indexOf, reverse
Performance Comparison

• Same output
  – padChars(5, "x") returns "xxxxx" in both cases

• String version
  ```java
  public static String padChars1(int n, String orig) {
      String result = "";
      for(int i=0; i<n; i++) {
          result = result + orig;
      }
      return(result);
  }
  ```

• StringBuilder version
  ```java
  public static String padChars2(int n, String orig) {
      StringBuilder result = new StringBuilder("");
      for(int i=0; i<n; i++) {
          result = result.append(orig);
      }
      return(result.toString());
  }
  ```
Wrap-Up
Summary

• **Stretchable lists: ArrayList & LinkedList**
  – Different performance characteristics
  – Declare variables to be of type List

• **Generics let you avoid tedious typecasts**
  – List<String> = new ArrayList<String>();

• **HashMap supports large lookup tables**
  – someTable.put(key, value);
  – SomeType value = someTable.get(key);

• **More**
  – Java stole printf from C. Yay!
  – Varargs provide for flexible argument lists
  – Use StringBuilder for repeated string concatenation in loops
  – Return results; don’t change data secretly
    • Except when doing so causes performance problems