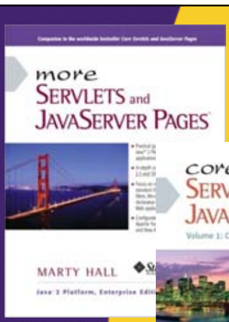
© 2013 Marty Hall

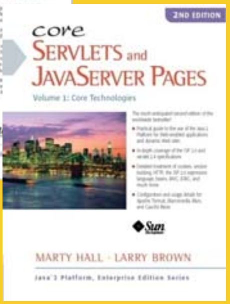
Java Programming: Syntax and Utilities II

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

Customized Java EE Training: http://courses.coreservlets.com/
Java, JSF 2, PrimeFaces, HTML5, JSP, Ajax, jQuery, Spring, Hibernate, RESTful Web Services, Hadoop, Android.
Developed and taught by well-known author and developer. At public venues or onsite at *your* location.

© 2013 Marty Hall





For live Java-related training,
see http://courses.coreservlets.com/
or email hall@coreservlets.com.

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.

- Courses developed and taught by Marty Hall
  - JSF 2, PrimeFaces, servlets/JSP, Ajax, jQuery, Android development, Java 7 or 8 programming, custom mix of topics
  - Courses available in any state or country. Maryland/DC area companies can also choose afternoon/evening courses.
- Courses developed and taught by coreservlets.com experts (edited by Marty)
  - Spring, Hibernate/JPA, GWT, Hadoop, HTML5, RESTful Web Services

Contact hall@coreservlets.com for details

Topics in This Section

- **Data structures**
  - ArrayList
  - LinkedList
  - HashMap
- **Using classes and methods that support generic types**
- **Defining your own classes and methods that support generic types**
- **printf**
- **varargs**
- **String vs. StringBuilder**

© 2013 Marty Hall

Lists and Generics

Customized Java EE Training: http://courses.coreservlets.com/
Java, JSF 2, PrimeFaces, HTML5, JSP, Ajax, jQuery, Spring, Hibernate, RESTful Web Services, Hadoop, Android.
Developed and taught by well-known author and developer. At public venues or onsite at *your* location.

Lists

- **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()`**

ArrayList Example

```
import java.util.*;  // Don't forget this import

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

Before Java 7, you had to repeat the type here, using new ArrayList<String>() instead of new ArrayList<>()

This tells Java your list will contain only strings. More on this in a few slides in section on generics.

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

	Array with Copying (`ArrayList`)	List of Pointers (`LinkedList`)
Insert at beginning	O(N)	O(1)
Insert at end	O(1) if space O(N) if not O(1) amortized time	O(1)
Access Nth Element	O(1)	O(N)

Hang on to your hats! We will walk carefully through what this all means in the lecture. But the whole reason that Java supplies both LinkedList and ArrayList is because of the different performance characteristics, so understanding this slide is the key to deciding which class to use in real life.

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");

ArrayList Example: Explicit Typecasts (Java 1.4 and Earlier)

```
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);
    }
  }
}
```

ArrayList Example: Generics (Java 7)

```
import java.util.*;

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

This tells Java your list will contain only strings. Java will check at *compile* time that all additions to the list are Strings. You can then use the simpler looping construct because Java knows ahead of time that all entries are Strings.

ArrayList Example: No Diamond Operator (Java 5 and 6)

```
import java.util.*;

public class ListTest3 {
  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);
    }
  }
}
```

Before Java 7, you had to use <String> instead of <> here. In Java 7 and later, the compiler will do type inferencing. But you still need List<String> in the variable declaration.

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

Old	New

`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);`

© 2013 Marty Hall

Maps
(Also called "Lookup Tables" or "Associative Arrays")

Customized Java EE Training: http://courses.coreservlets.com/
Java, JSF 2, PrimeFaces, HTML5, JSP, Ajax, jQuery, Spring, Hibernate, RESTful Web Services, Hadoop, Android.
Developed and taught by well-known author and developer. At public venues or onsite at *your* location.

HashMap

- **HashMap provides simple lookup table**
  - Use "put" to store data

```
Map<String,Person> employees =
   new HashMap<String,Person>();
Person p1 = new Person("a1234", "Larry", "Ellison");
employees.put(p1.getEmployeeId(), p1);
```

The table keys will be Strings; the associated values will be Persons.

  - Use "get" to retrieve data

```
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

HashMap Example: Finding State Abbreviations Based on State Names

```
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" }, ….
}
```

HashMap Example: Finding State Abbreviations Based on State Names

```
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

```
      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);
      }
    }
  }
```

© 2013 Marty Hall

Building Genericized Methods and Classes

Customized Java EE Training: http://courses.coreservlets.com/
Java, JSF 2, PrimeFaces, HTML5, JSP, Ajax, jQuery, Spring, Hibernate, RESTful Web Services, Hadoop, Android.
Developed and taught by well-known author and developer. At public venues or onsite at *your* location.

Generic Classes and Methods

- **Big idea**
  - Even beginning Java programmers need to know how to *use* classes that support generics:
    - List<Employee> workers = …;
    - Map<String, Employee> workerDatabase = …;
  - Intermediate Java developers should also to be able to *define* classes or methods that support generics.
- **Syntax**
  - If you put variables in angle brackets in the class or method definition, this tells Java that future uses of those variables refer to types, not instances.
    - public class ArrayList<E> { … }
    - public static <T> T randomElement(T[] array) { … }

Example: Generic Method

```
public class RandomUtils {
  private static Random r = new Random();

  public static int randomInt(int range) {
    return(r.nextInt(range));
  }

  public static int randomIndex(Object[] array) {
    return(randomInt(array.length));
  }

  public static <T> T randomElement(T[] array) {
    return(array[randomIndex(array)]);
  }
}
```

In rest of method, T refers to a type

This says that the method takes in an array of T's and returns a T. For example, if you pass in an array of Strings, you get out a String; if you pass in an array of Employees, you get out an Employee. No typecasts required in any of the cases.

Example: Generic Class (Simplified)

```
public class ArrayList<E> {

  public E get(int index) { ... }

  ...
}
```

In rest of class, E refers to a type

This says that get returns an E. So, if you created ArrayList<Employee>, get returns an Employee. No typecast required.

This is a highly simplified version of the real java.util.ArrayList class.
That class implements multiple interfaces, and the generic support comes from the interfaces.

© 2013 Marty Hall

printf

Customized Java EE Training: http://courses.coreservlets.com/
Java, JSF 2, PrimeFaces, HTML5, JSP, Ajax, jQuery, Spring, Hibernate, RESTful Web Services, Hadoop, Android.
Developed and taught by well-known author and developer. At public venues or onsite at *your* location.

Formatted 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 numbers)
- **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://docs.oracle.com/javase/7/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)
  - Using \% (wrong) instead of %% (right) if you want a literal percent sign

Printf Formatting Options

	Stands For	Options	Example
%s	**String.** Can output any data type. If arg is Object, toString is called.	%*width*s Gives min num of chars. Spaces added to left if needed.	printf("%8s", "Hi") outputs " Hi"
%d	**Decimal.** Outputs whole number in base 10. Also %x and %o for hex and octal.	%*width*d %,*width*d Gives min width; inserts commas.	printf("%,9d", 1234) outputs " 1,234"
%f	**Floating point.** Lets you line up decimal point and control precision.	%*width*.*precision*f %,*width*.*precision*f width includes comma and decimal point.	printf("%6.2f", Math.PI) outputs " 3.14"

%t*x*	**Time (or date).** %tA for day, %tB for month, %tY for year, and many more.	Date now = new Date();
		printf("%tA, %tB ,%tY", now, now, now)
		outputs
		"Thursday, November 17, 2005"	

%n	Outputs OS-specific end of line (linefeed on Linux, CR/LF pair on Windows)		

Printf Example: Controlling Width and Precision

```
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); }
}
```

© 2013 Marty Hall

Varargs

Customized Java EE Training: http://courses.coreservlets.com/
Java, JSF 2, PrimeFaces, HTML5, JSP, Ajax, jQuery, Spring, Hibernate, RESTful Web Services, Hadoop, Android.
Developed and taught by well-known author and developer. At public venues or onsite at *your* location.

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(someArrayOfInts)
- **Use sparingly**
    - You usually know how many arguments are possible

Varargs: Example

```
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));
  }
}
```

© 2013 Marty Hall

StringBuilder

Customized Java EE Training: http://courses.coreservlets.com/
Java, JSF 2, PrimeFaces, HTML5, JSP, Ajax, jQuery, Spring, Hibernate, RESTful Web Services, Hadoop, Android.
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 (bad if *n* can be large)**

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

- **StringBuilder version (good)**

```
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());
}
```

© 2013 Marty Hall

Wrap-Up

Customized Java EE Training: http://courses.coreservlets.com/
Java, JSF 2, PrimeFaces, HTML5, JSP, Ajax, jQuery, Spring, Hibernate, RESTful Web Services, Hadoop, Android.
Developed and taught by well-known author and developer. At public venues or onsite at *your* location.

Summary

- **Stretchable lists: ArrayList and LinkedList**
  - Different performance characteristics
  - Declare variables to be of type List, not ArrayList or LinkedList
- **Generics let you avoid tedious typecasts**
  - List<SomeType> entries = new ArrayList<SomeType>();
    - Java 7: List<SomeType> entries = new ArrayList**<>**();
- **HashMap supports large and fast 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 string concatenation in loops

© 2013 Marty Hall

Questions?

JSF 2, PrimeFaces, Java 7 or 8, Ajax, jQuery, Hadoop, RESTful Web Services, Android, HTML5, Spring, Hibernate, Servlets, JSP, GWT, and other Java EE training. Also see the Java 8 tutorial and general Java programming tutorial.

Customized Java EE Training: http://courses.coreservlets.com/
Java, JSF 2, PrimeFaces, HTML5, JSP, Ajax, jQuery, Spring, Hibernate, RESTful Web Services, Hadoop, Android.
Developed and taught by well-known author and developer. At public venues or onsite at *your* location.