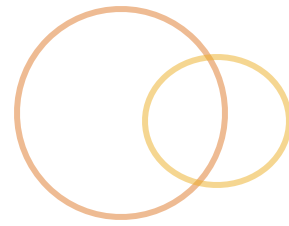


Java JDK Enhancements

Basic Language Enhancements



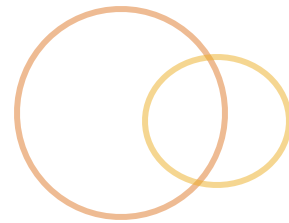
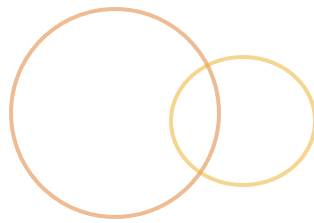
Presentation Topics



In this presentation we will cover:

- ⦿ Static imports
- ⦿ Autoboxing
- ⦿ For-each loops
- ⦿ Varargs

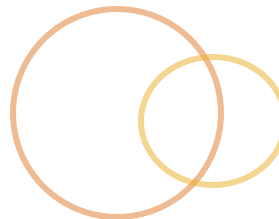
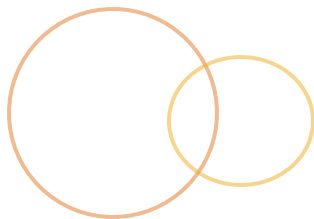
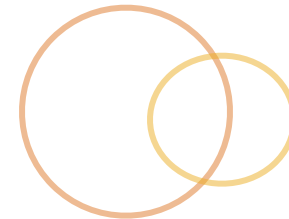
Objectives



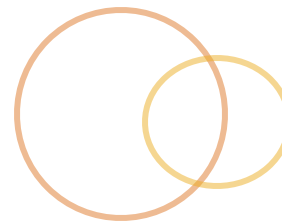
When you are done, you should be able to:

- 🕒 Identify three basic language enhancements
- 🕒 Describe when two enhancements are applicable
- 🕒 Create a basic prototype using all four basic language enhancements

Static Imports



Static Imports



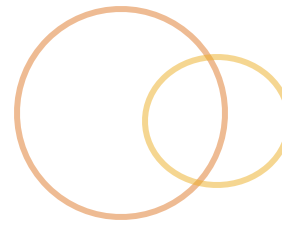
What are they?

- Mechanism for importing static variables and methods
- Very similar to standard import syntax

Why do they exist?

- Simplify access to static variables and methods in code
- Restore cohesion problem found in “work-around” solutions (aka – implementing an interface to gain access to static variables)

Static Imports [cont.]



⦿ How do they work?

- ⦿ Like normal import mechanism
 - ⦿ Development-time short-cut
 - ⦿ Compiler converts short-cuts into fully qualified names
- ⦿ In static imports
 - ⦿ Compiler converts “static” short-cuts into fully qualified names

Working With Static Imports



- ◎ Two types of static import
 - ◎ Single static import declaration
 - ◎ Static “on-demand” import declaration
- ◎ Look similar to . . .
 - ◎ Single type import declaration
 - ◎ “On-demand” type import declaration
- ◎ . . . but work a little different
 - ◎ Single static import - imports single static variable or function
 - ◎ “On-demand” static import - imports all static variables and functions

Static Import Example [old way]



```
1 package examples.staticimport;
2
3 /**...*/
7 class StaticImport {
8
9     public static void main(String [] args) {
10         double circumference = 7.7;
11         double diameter = circumference * Math.PI;
12         double roundedDiameter = Math.round(diameter);
13         System.out.println("The diameter of the circle is: " + diameter);
14         System.out.println("The rounded diameter is: " + roundedDiameter);
15     }
16
17 }
18
```


Single Static Import Example



```
1 package examples.staticimport;
2
3 import static java.lang.Math.PI;
4 import static java.lang.Math.round;
5
6 /**...*/
11 class SingleStaticImport {
12
13     public static void main(String [] args) {
14         double circumference = 7.7;
15         double diameter = circumference * PI;
16         double roundedDiameter = round(circumference);
17         System.out.println("The diameter of the circle is: " + diameter);
18         System.out.println("The rounded diameter of the circle is: " + diameter);
19     }
20
21 }
22
```

On-Demand Static Import Example

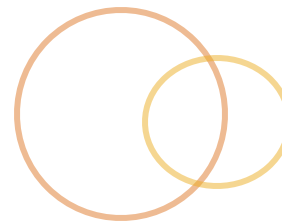


```
1 package examples.staticimport;
2
3 import static java.lang.Math.*;
4
5 /**...*/
10 class OnDemandStaticImport {
11
12     public static void main(String [] args) {
13         double circumference = 7.7;
14         double diameter = circumference * PI;
15         double roundedDiameter = round(circumference);
16         System.out.println("The diameter of the circle is: " + diameter);
17         System.out.println("The rounded diameter is: " + roundedDiameter);
18     }
19
20 }
21
```

Static Import Best-Practices



- ◎ Be aware
 - ◎ Name-space collisions can occur
 - ◎ Code can be hard to read
- ◎ Be specific
 - ◎ Consider avoiding wildcard notation
 - ◎ Use “optimize imports” functionality of IDE
- ◎ Avoid abuse
 - ◎ Perform proper OOAD anytime you create a static
- ◎ Refactor old code



🕒 **Description:**

Create a stand-alone Java application called `Mixer`. `Mixer` can accept any number of command line arguments. If `Mixer` receives 3 or less arguments, `Mixer` should sort the arguments using `Arrays.sort` and print the results. If `Mixer` receives more than 3 arguments, `Mixer` should sort the arguments, count the frequency of each argument, and print the argument and its frequency in sorted order.

🕒 **Duration:** 15 minutes

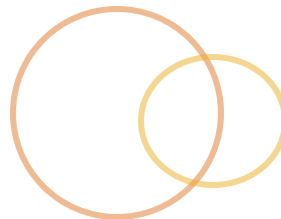
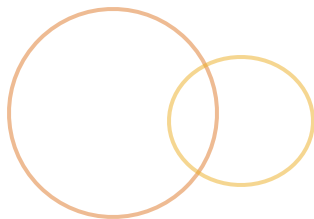
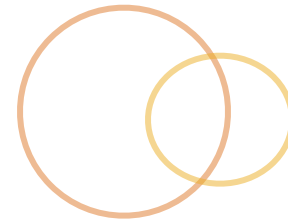
Static Import Lab Solution



```
44 private static Map getFrequencyMap(String[] args) {  
45     Map returnMap = new TreeMap();  
46     List list = asList(args);  
47  
48     for(int i=0;i<args.length;i++) {  
49         if(!returnMap.containsKey(args[i])) {  
50             int freq = frequency(list, args[i]);  
51             returnMap.put(args[i], new Integer(freq));  
52         }  
53     }  
54  
55     return returnMap;  
56 }  
57
```

Solution source: `labs.solutions.staticimport.Mixer.java`

Autoboxing





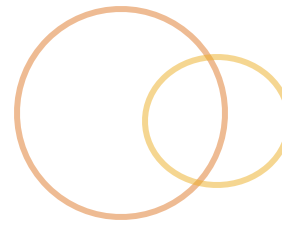
Terminologies

- Boxing - converting primitive to reference
- Unboxing - converting reference to primitive
- Auto - short for automatic; opposite of manual

What is it?

- General term describing automatic boxing and unboxing of data
- Should be called Auto-boxing and Auto-unboxing

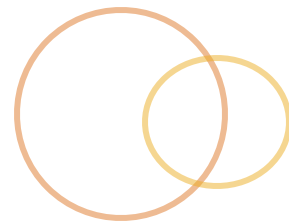
Autoboxing [cont.]



Why does it exist?

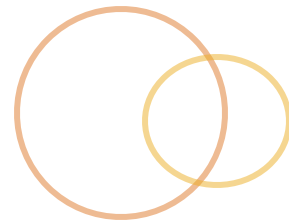
- Simplify data conversion process from primitive to reference and reference to primitive
- Simplify code (ease of development)
- Hide “side-step”
- Be more “modern”

Autoboxing Support



Primitive Type	Reference Type
boolean	<code>java.lang.Boolean</code>
byte	<code>java.lang.Byte</code>
short	<code>java.lang.Short</code>
int	<code>java.lang.Integer</code>
long	<code>java.lang.Long</code>
float	<code>java.lang.Float</code>
double	<code>java.lang.Double</code>
char	<code>java.lang.Character</code>

Autoboxing [cont.]



⦿ How does it work?

- ⦿ According to lang spec, occurs at run-time
- ⦿ Utilizes wrapper class functionality

Simple Autoboxing Example



```
1  package examples.autobox;  
2  
3  /**...*/  
8  public class SimpleAutoboxingExample {  
9  
10     public static void main(String[] args) {  
11         //box the int value of 32 into  
12         //a Integer reference  
13         Integer x = 32;  
14  
15         //unbox the Integer reference into  
16         //a int value and multiply by 2  
17         int y = x * 2;  
18  
19         //no boxing going on here  
20         System.out.println("The value of y is: " + y);  
21     }  
22 }  
23
```

Collection Autoboxing Example



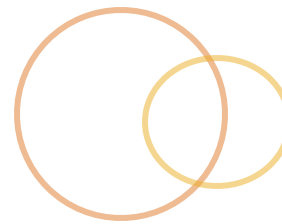
```
1  package examples.autobox;
2
3  +import ...
6
7  +/** ... */
12 public class WordCount {
13
14  - public static void main(String[] args) {
15      Map wordMap = new HashMap();
16      for(int x=0;x<args.length;x++) {
17          int wordCount = 1;
18          if(wordMap.containsKey(args[x])) {
19              wordCount = ((Integer) wordMap.get(args[x])) + 1;
20          }
21          wordMap.put(args[x], wordCount);
22      }
23
24      Iterator itr = wordMap.keySet().iterator();
25      while(itr.hasNext()) {
26          String key = (String) itr.next();
27          int value = (Integer) wordMap.get(key);
28          System.out.println(key + " has " + value + " occurrences");
29      }
30
31  - }
32  }
33
```

Autoboxing Best Practices



- ◎ Be aware of performance implications
 - ◎ Avoid using boxing / unboxing when possible
 - ◎ Only use when there is a type mismatch
 - ◎ Don't get lazy
- ◎ Know the edge cases
 - ◎ `NullPointerException`
 - ◎ `OutOfMemoryError`

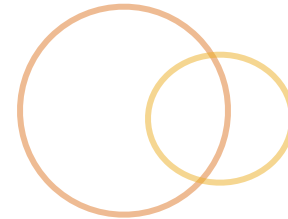
Autoboxing Lab



🕒 **Description:** Use the `Mixer` application you wrote in the static import lab as your starting point. Apply autoboxing concepts to streamline the code that put the frequency count into the `Map`. Apply autoboxing concepts to streamline the code when you get the frequency count from the `Map`. Changing the `Mixer` to use autoboxing should not effect the results of the application.

🕒 **Duration:** 15 minutes

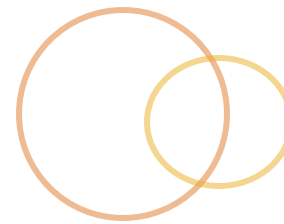
For Each Loop



Working with the enhanced for loop



For-each Loop



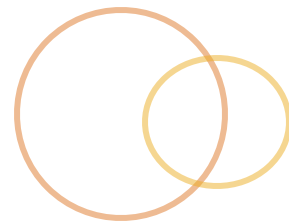
What is it?

- A variation of a for loop
- Simplified notation targeted at collections and arrays

Why does it exist?

- Remove redundant iteration code
- Simplify iteration over collections

For-each Loop [cont.]



⦿ How does it work?

- ⦿ Functions like a for loop . .
 - ⦿ Iterate over collection
 - ⦿ Access each collection element individually
- ⦿ . . but has a different syntax
 - ⦿ Has an initialization “clause” and “expression” clause
 - ⦿ Initialization clause holds current element in collection
 - ⦿ Expression clause defines collection
 - ⦿ Doesn't have a “test” clause or “increment” clause
 - ⦿ Clauses separated by : instead of ;
- ⦿ Translated into a formal for loop at compile time

Simple For-each Loop Example



```
1  package examples.foreach;
2
3  /** ... */
10 public class SimpleForEach {
11
12     public static void main(String[] args) {
13         //initialization clause - String s
14         //expression clause - args
15         for(String s : args) {
16             System.out.println(s);
17         }
18     }
19 }
20
```

Limitations of For-each Loop



⦿ Limited type support

- ⦿ Supports arrays

- ⦿ Supports `java.lang.Iterable`

 - ⦿ Collections support `Iterable` through class hierarchies

 - ⦿ `Iterable` provides `java.util.Iterator` to the for-each loop

⦿ Limited insight

- ⦿ No way to determine “where am I” during iteration

- ⦿ No access to iterator or index

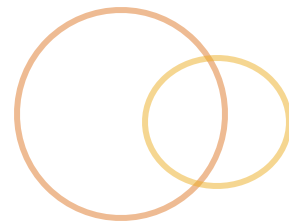
⦿ All clauses are required - no “endless loop” abilities

Iterable For-each Loop Example



```
1  package examples.foreach;
2
3  import java.util.Arrays;
4  import java.util.List;
5
6  /**
7   * The following illustrates using the for-each
8   * loop with a collection through the Iterable
9   * interface.
10 */
11 public class IterableForEachExample {
12
13     public static void main(String[] args) {
14         //convert the array into a list
15         List argList = Arrays.asList(args);
16
17         //iterate over the list
18         for(Object arg : argList) {
19             System.out.println(arg);
20         }
21     }
22 }
23
```

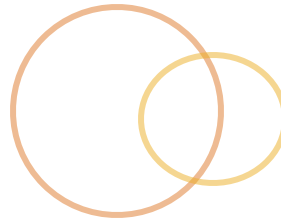
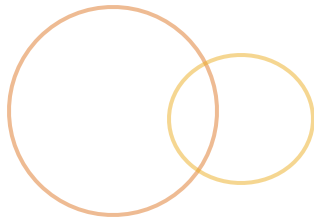
For-each Loop Lab



🕒 **Description:** Use the `Mixer` as your starting point. Where applicable, convert the for-loops in the `Mixer` into the for-each loop syntax. The modification should not change the program's results.

🕒 **Duration:** 10 minutes

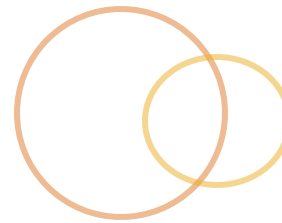
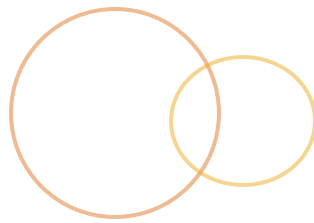
Variable Argument Lists



Variable Argument Lists



- ◎ What is a Variable Argument List?
 - ◎ Associated with a method
 - ◎ Makes method argument list dynamic in length
 - ◎ Requires modification of method signature
 - ◎ Allows dynamic number of arguments to be passed into a method



Why does it exist?

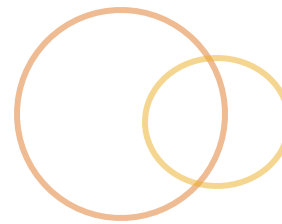
- Simplifies passing flexible number of arguments
- Typical argument list notation is inflexible
 - Certain scenarios require flexibility
 - Flexibility was supported through an `Object[]` argument
- `Object[]` support was cumbersome
 - Required developer to declare method with `Object[]` argument
 - Required developer to convert parameters into an array before passing

Varargs Example [old way]



```
1 package examples.varargs;
2
3 /**...*/
13 public class VarArgsOldWay {
14
15     public static void main(String[] args) {
16         String name = "John Doe";
17         String book1 = "Hooked On Java";
18         String book2 = "The Java Language Specification";
19         //convert arguments into array
20         String [] titles = {book1, book2};
21         //pass arguments as array
22         listBooks(name, titles);
23     }
24
25     private static void listBooks(String name, String[] titles) {
26         System.out.print(name + " likes: ");
27         for(int i=0;i<titles.length;i++) {
28             System.out.print("\"" + titles[i] + "\"");
29             if(i+1 < titles.length)
30                 System.out.print(",");
31         }
32         System.out.flush();
33     }
34 }
35
```

Using Varargs



⦿ How does it work?

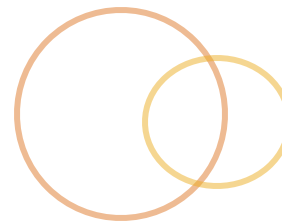
- ⦿ Change method signature to support varargs
 - ⦿ Include ***ellipse*** notation
 - ⦿ Must be last argument in argument list
- ⦿ Pass varargs either as:
 - ⦿ An array of objects (like old way)
 - ⦿ Like normal arguments (comma separated)
- ⦿ Varargs automatically converted
 - ⦿ Signature converted to support []
 - ⦿ Arguments converted into an array
 - ⦿ All performed by compiler
- ⦿ Work with varargs like any other array

Simple varargs Example



```
1  package examples.varargs;
2
3  /**...*/
11 public class VarArgsNewWay {
12
13     public static void main(String[] args) {
14         String name = "John Doe";
15         String book1 = "Hooked On Java";
16         String book2 = "The Java Language Specification";
17         //pass arguments as arguments
18         listBooks(name, book1, book2);
19     }
20
21     private static void listBooks(String name, String... titles) {
22         System.out.print(name + " likes: ");
23         for(int i=0; i<titles.length; i++) {
24             System.out.print("\"" + titles[i] + "\"");
25             if(i+1 < titles.length)
26                 System.out.print(", ");
27         }
28         System.out.flush();
29     }
30 }
31
```

Varargs Lab

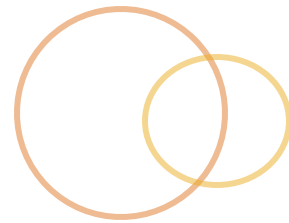
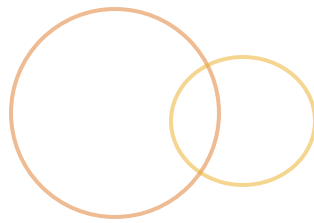


🕒 Description:

Use `Mixer` as your starting point. Create a method called `print`. `print` should be a variable argument method accepting `Object`. Change `Mixer` to use the `print` method. If it receives two values, print the results as `name=value`. Otherwise, print the values as a comma-separated list.

🕒 Duration: 20 minutes

Summary



Four basic language enhancements

- ◎ **Static Imports** - simplify referencing static members in source code
- ◎ **Autoboxing** - simplify boxing / unboxing operations
- ◎ **For-each loops** - simplify iterating over a collection
- ◎ **Varargs** - simplified mechanics for creating variable length argument list