Java 5 Features, What's in it for you?

Venkat Subramaniam

venkats@agiledeveloper.com

http://www.agiledeveloper.com/download.aspx

Abstract

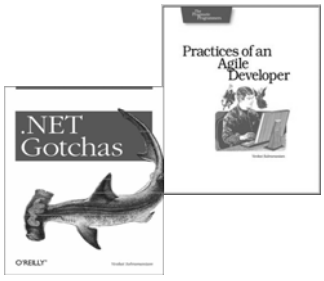
Abstract A number of new features have been introduced in Java. What benefit do these features offer you. Are there issues with using these features. For instance, when should you use annotation? The objective of this presentation is not simply to introduce you to the features, but to the effective use of these as well.

We will take a close look at a number of features that you will be expected to know well when you program using Java 5.

About the Speaker *Dr. Venkat Subramaniam*, founder of Agile Developer, Inc., has trained and mentored thousands of software developers in the US, Canada, and Europe. He has significant experience in architecture, design, and development of software applications. Venkat helps his clients effectively apply and succeed with agile practices on their software projects, and speaks frequently at conferences.

He is also an adjunct faculty at the University of Houston (where he received the 2004 CS department teaching excellence award) and teaches the professional software developer series at Rice University School of continuing studies.

Venkat has been a frequent speaker at No Fluff Just Stuff Software Symposium since Summer 2002.

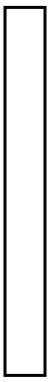


Java 5 Features

- **New Features**
- Autoboxing
- For-each
- enum
- Varargs
- Annotation
- Static Import
- Generics
- Other features
- Conclusion

New Features

- Java 5 has introduced several new language and API features
- Most of the change are welcome changes
- A few can be abused
- One is a disaster



Java 5 Features

- New Features
- **Autoboxing**
- For-each
- enum
- Varargs
- Annotation
- Static Import
- Generics
- Other features
- Conclusion

Citizens of Java

- Java has two classes of citizens
  - Objects and primitives
- You couldn't mix them together
- What if you want to write a generalized API to receive different types?
  - Like Invoke method or even collection classes
- You had to wrap the primitive into an object
  - Predefined objects like Integer (for int), Double (for double), etc.
- You have to wrap on send and cast on receive

Dealing with differences

- Leads to code clutter
- More work for you
- Some what unnatural
- Confuses the heck out of novices

AutoBoxing and Unboxing

- Allows you to treat primitives and corresponding object types interchangeably
- You some what forget the difference as you work with the code
- You can send an int where an Integer is expected
- You may assign an Integer to an int
- The wrapping, unwrapping is done for you as autoboxing and autounboxing

+/-

- Realize that the conversion is happening still under the hood
  - Has performance consequences
- What if Integer reference is null and you try to assign it to int
  - NullPointerException
- What does == mean now?
  - On Integer it is identity comparison and on int it is value based comparison, use caution when you mix
- Use it where performance is not critical

Java 5 Features

- New Features
- Autoboxing
- **for-each**
- enum
- Varargs
- Annotation
- Static Import
- Generics
- Other features
- Conclusion

Looping

- How can you loop?
  - It depends on what you are looping over
- You can use the good old method:

```
for(int index = 0; index < arr.length;
index++) {…}
```

- Or you may use:

```
for(
    Iterator iter = c.iterator();
    iter.hasNext(); ) {… iter.next()…}
```

- Again, we have lived with this clutter
- How about something simpler?

for-each

- How about reading it as
  for each element X in collection col …
- Syntactically this may look like
  foreach(X element in col) // WRONG
- But this would lead to two problems
  - Would introduce two new keywords
  - May step on the toe of some legacy code
- Smartly avoided with an uglier syntax ☺

```
for(X element : col) { …
    // Each pass through loop, element refers to
    // an element in the collection col
}
```

+/-

- It does look simpler, elegant
- But you can't use all the time
- You can't use it if you need access to the iterator or index it hides
  - like to remove an element or set a value in a collection using an index
- For you to use for-each, the collection must implement a Iterable interface
  - This has one method iterator() that returns the iterator

Java 5 Features

- New Features
- Autoboxing
- For-each
- **enum**
- Varargs
- Annotation
- Static Import
- Generics
- Other features
- Conclusion

What it was?

- Type whose legal value is from fixed set of constants
- 'C' enum was omitted from Java
- You want to define a select set of possible values
- You wrote

```
public static final int COFFEESIZE_SMALL = 1;
public static final int COFFEESIZE_MEDIUM = 2;
public static final int COFFEESIZE_LARGE = 3;
```

- Joshua Bloch has talked about why this is bad in "Effective Java"

What's wrong?

- Item #21 "Replace enum constructs with classes" in Effective Java
  - Not typesafe
  - No namespace (how can you have two enumerations within same scope?)
  - Brittle – errors result from adding new ones, may conflict with existing ones
  - Not printable – how to nicely print a value of enum?
- Solution? – typesafe enum pattern

typesafe enum pattern

- Joshua's recommendation:

```
public class CoffeeSize
{
   private final String theName;
   private CoffeeSize(String name)
        { theName = name; }
   public String toString()
        { return theName; }
   public static final CoffeeSize SMALL
                        = new CoffeeSize("Small");
   public static final CoffeeSize MEDIUM
                        = new CoffeeSize("Medium");
   public static final CoffeeSize Large
                        = new CoffeeSize("Large");
}
```


Advantages

- No way for you to create objects of this class
  - You only have objects you have exported
- You can't extend this class since the constructor is private
- Provide compile-time type safety
  - Except you may pass a null
- Provides namespace for defining multiple enums with names across enums
  - Like *Small* CoffeeSize and *Small* ShirtSize
- You can add constants without breaking client code
- Printable as you desire by overriding toString()
- Can add methods
- You can add the enumerations to a collection if you want to get a list of all values

Minor Disadvantages

- Hard to aggregate like Small | Large
- Can't use switch on them – you need to resort to if/else statements
- You are loading classes at runtime – may be an issue for resource constrained devices
- Pretty minor concerns, so why not use it?

- Verbose, lot of work to write
- Wouldn't it be nice if this is done for us?

Enum

- Looks like C/C++ support - only better

  enum CoffeeSize { SMALL, MEDIUM, LARGE; }
- Behind the scene enum type generation
- They are comparable and serializable
- Has values() method that returns array of all values of enum type in order of declaration
- Can use switch if you like

Enhancing enum

- You may add data and behavior to an enum
- You may write a constructor that takes arguments and declare const values with values for the parameters
- Constant specific methods
  - You may declare a method abstract in the enum
  - May override the method in each constant value of enum

enum specific classes

- java.util.EnumSet
  - High performance implementation for enums
  - Stored internally as bit vector
  - Provides iterator over range of enum values
  - range() method
  - of() method creates aggregation of enum values
- java.util.EnumMap
  - Allows you to map enum to values

+/-

- enum are very powerful
- Provide type safety

- Can be confusing however if you do some of the fancy things with it

Java 5 Features

- New Features
- Autoboxing
- For-each
- enum
- **Varargs**
- Annotation
- Static Import
- Generics
- Other features
- Conclusion

Sending variable number of args

- Two reasons you forced you to send an array to a method
  - You wanted to pass a primitive type by reference
  - You wanted to send variable number of arguments
- Not elegant, how about something simpler

varargs

- varargs is a sugar coating that hides the creation of array for you
- public static object max(

  Object... arguments)
- Ellipsis is used to say that the argument may be passed as an array or a sequence of values
  - Only the trailing argument can have ellipsis

Nice work

- Ellipsis in C++ is horrible
  - Hard to use
  - Leads to significant issues with determining the type correctly
- Java's implementation is a lot more civilized
- You may mix varargs with autoboxing as well
- Reflection API methods like invoke accept varargs
- System.out.printf

+/-

- Use it when it makes sense to provide flexibility of receiving variable number of arguments
- You loose some compile time type safety
- If you are debating between say 3 and 4 parameters, you may consider overloading instead
- If you are using varargs, then don't overload – leads to confusion

Java 5 Features

- New Features
- Autoboxing
- For-each
- enum
- Varargs
- **Annotation**
- Static Import
- Generics
- Other features
- Conclusion

Metadata

- You want to communicate certain intent in your code
  - Like serializable, Cloneable, etc.
- Use of interface and inheritance is not the smartest idea
- Tagging interfaces bloat code
  - Intent is not clear, more of a work around, using that inheritance hammer
  - Can't help with anything more granular than methods – remember transient, you have to invent keywords, not something developers can do
- How about side files like deployment descriptors
  - Error prone, messy, gets too complex, hard to keep up with

Metadata Facility

- Annotation brings ability to annotate or color code for general purpose
- Annotation types can be defined
- Annotation declaration can be placed
- Annotation can be identified programmatically

Annotation Type Declaration

- Use @interface to declare
- Each method declares an element
  - No parameters
  - No throws
  - No implementation
  - Return type must be primitives, String, Class, enum, annotation, or array of these
  - Can have default values
- Marker annotation has no methods
- If single element, name it value

Annotation Declaration

- Used like modifiers
- Convention to place it before the other modifiers like public, static, etc.
- Has parenthesized list of name value pair
- For marker annotation, no parenthesis
- For single valued annotation, no need for the “elementName =“ – simply provide the value

Meta-Annotation

- You want to say a few things about the annotation itself
  - Where it can be used, how, etc.
- Retention says whether the VM retains it for reflective access at runtime
- Target’s ElementType specifies where the enum can be used: Class, Method, etc.

Fetching Annotation Info

- Reflection API can be used to explore the Annotation details at runtime programmatically
- isAnnotationPresent()

apt

- Annotation Processing Tool
- Command line utility for annotation processing
- Support API with reflection
- Helps with development of code by offering code generation and compilation

+/-

- Annotation very powerful
- Better than using tagging interfaces
- If intent is to affect documentation, use javadoc
- Otherwise consider use of annotation
- Learn if annotation is the right thing for what you want to specify, however

Java 5 Features

- New Features
- Autoboxing
- For-each
- enum
- Varargs
- Annotation
- **Static Import**
- Generics
- Other features
- Conclusion

How do we use static methods?

- To use static methods, we prefix the call with class name
  - Runtime.getRuntime()
- Disadvantage: The code is readable (pun intended)
- Wouldn't it be cool to simply call
  getRunTime()
  and let the programmer scratch the head asking where the heck this came from?

Static Import can do that for you

- Allows you to call static methods without prefixing with class names
- You import, hum, statically, the type that has the static members

  import static java.lang.System.out;

or

  import static java.lang.Runtime.;*

- Very similar in rules/behavior as regular import is

+/-

- You may use it when you need frequent access to a select set of static members
- Makes the code less readable in my pinion
- Some organization do not encourage use of import – this makes it worse
- May lead to job security for the writer?

Java 5 Features

- New Features
- Autoboxing
- For-each
- enum
- Varargs
- Annotation
- Static Import
- **Generics**
- Other features
- Conclusion

Generics

- Remember the good old Templates in C++?
- Java went the route of using Object as generic type
  - Problem is when you pull some thing out of a collection, how do you call methods on it?
  - Only after casting it to the correct type right
  - Much worst if you are dealing with primitive types
    - These have to be boxed and unboxed
- Having collections that are type safe will eliminate this issue
  - Back to what C++ originally provided ☺

Need?!

- This is highly debatable
- First question is do we really need a type safe language
  - What about dynamically typed languages
- If we used dynamically typed languages, then we do not really care about generics!
- But then, we are talking about Java here
- So, how do we solve the issues with such a strongly typed language

+ / -

- Easier to write
- Intended to provide type safety
- Nice features like extends and super to control the type for parametrized type
- No change to JVM or byte code for generics
- Good news – compatible with legacy code
- Bad news – you loose type safety
- More about generics in
  - Good, Bad, and Ugly of Java Generics talk
  - Three part article at http://www.agiledeveloper.com/download.aspx

Java 5 Features

- New Features
- Autoboxing
- For-each
- enum
- Varargs
- Annotation
- Static Import
- Generics
- **Other features**
- Conclusion

Other Features

- StringBuilder
  - StringBuffer eliminates object creation overhead, but has synchronization overhead
  - StringBuilder removes that overhead
- Client vs. Server side differences in garbage collection, more adaptive collection
- Improved Image I/O for performance and memory usage
- Reduced application startup time and footprint using shared archive
- Enhancement to Thread Priority
- Ability to get stack trace for a thread or all threads
- UncoughtExceptionHandler on a Thread
- Improved error reporting on fatal exceptions
- System.nanoTime() for nanoseconds granularity for time measurements

Other Features...

- ProcessBuilder
  - Easier than Runtime.exec() to start process
- Formatter and Formattable provide ability to format output in printf like style
- Scanner for easier conversion to primitive types – based on regex
- java.lang.instrument allows byte code enhancement at runtime to instrument code
- Collections Framework has Queue, BlockingQueue, and ConcurrentMap interfaces and implementations. Some classes modified to implements new interfaces
- Reflection API supports annotation, enum. Class has been generified
- System.getenv() undeprecated!

Quiz Time

Java 5 Features

- New Features
- Autoboxing
- For-each
- enum
- Varargs
- Annotation
- Static Import
- Generics
- Other features
- **Conclusion**

Love-hate of features

- Autoboxing ☺
- For-each ☺
- enum ☺
- Varargs ☺
- Annotation ☺☹
- Static Import ☹
- Generics ☹☹☹….

References

1. http://java.sun.com

2. Download examples/slides from

http://www.agiledeveloper.com/download.aspx

Please fill out your evaluations!