Core Java

18. Future Directions in Java

Annotations

- Annotated code can save lots of boilerplate conditionals/constructors
- Used in new Java EE Web Services, Java Persistent API, Java Servlet API
- *@interface, @Overrides, @Target(METHOD), @Deprecated*
- Used by Annotation processors for generating related files/code
- Reflection Extensions to read them

Generics and Closures

- Notation of type-safe generics extended to runtime as well
- Helps detect malicious writes with casts
- So you can be sure that an *ArrayList* intended to take *Strings* won't take *Integers*
- Closure to confusion? Overload chunks of language constructions with your own implementation (somewhat equivalent to Operator overloading)

New I/O API

- Think of *Channel*s and *Buffer*s
- Supports memory mapping, non-blocking I/O
- Provides character-set encoders
- Overall better performance than *java.io*
- Used in Java EE servers today

Multiple Languages

- .NET isn't the only framework which has multiple languages
- Groovy, BeanShell, Jython, Rhino
- Implement multiple languages using same runtime
- Useful for prototyping/scripting

Superpackages

- A *Jar* of *Jars* is a *Jam*
- Used to pack multiple *JAR*s into a single Java Module (*JAM*)
- Specification for abstracting overall Module

Java ME

- Set of APIs for Mobile Java
- Limited subset of Java language
- More popular as a platform for Mobile applications
- Java in mobiles is now commonplace
- Great potential for development, Good availability of SDKs
- Open mobile platforms

Java EE 5 and Beyond

- Set of Enterprise APIs based on Java
- Distributed, Concurrent Programming
- Used to run transactions in various enterprises
- More APIs being pushed into Java SE for overall benefit

Java Community Process

- Participating in forming the Java language
- Democratic procedure
- *Java Specification Request*
- Vendor Neutrality: Opens API so everyone can benefit
- Lots of new *JSR*s eventually get approved and make it into Java SE/EE/ME

Questions?