for

San Antonio Java User Group

May 2010

# **Top 5 Main features in Java 7**

- Modularity
- Language Changes
- Multi Language Virtual Machine
- Garbage Collector
- New File I/O API

# **Modularity – What is the need?**

- \* Java SE has grown BIG
  - More and more functionality is added in Java SE
  - •Java SE the foundation of Java EE
- Underlying Implementation code is interconnected
- Class path Hell
  - No version management for JARs
  - No dependencies mechanism

# **Module – an Example**

module A @ 1.0

```
requires B @ 2.1;
        requires C @ 1.1;
module A;
package com.sample;
public class sample class
 public void sample()
```

module definition stored in moduleinfo.java

- -The sample class belongs to the module A version 1.0
- This class is public
- If class or one of its members or constructors is declared module, it will be accessible from a type that belongs to the same module

## **Language Changes**

- Additions to the Java Language itself
- Sun very much conservation to language additions
  - because language changes are permanent and cannot be removed
- No big features like support for closures
- Only simple and useful language changes in Java 7
- Under the Open Source Project Coin (signifies small changes)

# <u>Language Changes – Strings in switch</u>

```
String s = \dots;
switch(s)
        case "subbu":
                 System.err.println("It is subbu man!");
                 break;
        case "ryan":
                 System.err.println("It is ryan man!");
                   break;
        case "john":
        default:
                 System.err.println("Default");
                 break;
```

# **Language Changes – Multiple Exception Handling**

```
try
 //do someting;
catch(Exception1 e)
 handleException(e)
catch(SQLException e)
 handleException(e)
```

```
try
{
  //do someting;
}
catch(Exception1, Exception2 e)
{
  handleException(e)
}
```

# <u>Language Changes – Improved Type</u> <u>Inference</u>

Map<String, Integer> = new HashMap<String, Integer>();

Map<String, Integer> = new HashMap<>();

# <u>Language Changes – Elvis Operator</u>

### Standard example:

String s = mayBeNull?.toString() ?: "null";

### Auto-unboxing example:

```
Integer ival = ...; // may be null int i = ival ?: -1; // no NPE from unboxing
```

# <u>Language Changes – Elvis Operator</u>

### Standard example:

String s = mayBeNull?.toString() ?: "null";

### Auto-unboxing example:

```
Integer ival = ...; // may be null int i = ival ?: -1; // no NPE from unboxing
```

# Multi Language Virtual Machine

- Java 6 provided the capability to plug-in other languages in the VM
- Java 7 to be strengthened to make the other languages run even better in the JVM
- JVM to be optimized to make the language specific features run better (Ex: Closures)
- Provided by Open Source Project DaVinci

### **Limitation of current I/O API**

- •Deletion is not guaranteed. Need to do a check to determine if the file is deleted.
- •Operations on directory are not scalable and run on the parent thread
- •Polling will need to be done for changes on files

### New I/O APIs

- New File System API
- File Notifications
- Directory Operations
- Asynchronous I/0

## New I/O APIs - Example

```
Path searcPath = Paths.get("c:/sample");
final Path findFile = Paths.get("samplefile");
FileVisitor visitor = new SimpleFileVisitor()
        public FileVisitResult visitFile(Path file, BasicFileAttributes
attrs)
                 if(file.getName().startsWith(findFile)
                          //do something
Files.walkFileTree(searcPath, visitor);
```

## **Garbage First**

- •New Garbage Collector introduced in Java 7
- It is called the Garbage First Collector
- Memory split into multiple regions as opposed to 2 regions in the current version
- Quite predictable and provides greater through put for memory intensive applications
- Performs faster than the current parallel collectors.

## New I/O APIs - Example

```
Path searcPath = Paths.get("c:/sample");
final Path findFile = Paths.get("samplefile");
FileVisitor visitor = new SimpleFileVisitor()
        public FileVisitResult visitFile(Path file, BasicFileAttributes
attrs)
                 if(file.getName().startsWith(findFile)
                          //do something
Files.walkFileTree(searcPath, visitor);
```

Any Questions? Thank You