# Software Future

## Java 7 Ideas

- These are pretty speculative at this time. Java 7 might ship in early 2009 (this is just my guess)
- Some sort of closure system -- syntactic show way to make little bits of code to pass around
  - For example, something you could use for the code here: SwingUtilities.invokeLater( Runnable() { ..code.. } );
  - Maybe bits of code look like { int x => x * x } (takes in x, returns x*x )
  - e.g. new Thread( { => 6 } ) -- create a thread with a closure that just returns 6.
  - Under the hood, these are making something like inner classes, but with an easier syntax.
- Maybe get rid of "erasure" generic system -- generic types work more fully (at the cost of compatibility with old JVMs, while old source code will compile fine).
- Properties -- maybe get rid of the need to write trivial get/set methods ... they are automatically there based on some keyword, and you can write custom get/set code at any time if you wish.
- Better var declaration -- avoid repeating too much type info
  - {m := new HashMap<String, Integer>()}
  - Deduce the compile time type of m rather than making us type it in twice
- Superpackages -- give more control about what is effectively public vs. private between large bodies of code.

## Java on the Client

- Java applets/JWS are not widely popular
- Main problem: startup time too slow
- Other problem: many more people can write flash animations than write java code

## Java 6 "Consumer JRE"

- Release soon -- early 2008?
- Tricks to make JRE start fast -- rumors are that it works well (like Flash)
- Make the JRE download faster by breaking it into pieces

## JavaFX

- New experimental project (open source, like the rest of Java)
- JavaFX is supposed to work like Flash -- you can "draw" your app in a tool vs. having to write code. Makes simple/common web site type thing work easily
- JavaFX script ultimately will compile to bytecode, runs in the JRE (IMHO inventing yet another custom language, JavaFX Script, was dumb. The last thing I need is yet another syntax to learn.)
- Competes with Flash etc. -- making fancy graphical web sites
- Also supposed to work on phones
- Basically, a graphical, easy to program sibling of Swing
- JavaFX Problems: flash already exists and is hugely popular, and the JRE needs to prove that it can have super-fast startup
- JavaFX Advantages: Flash and Microsoft's Silverlight are totally proprietary. I'd much rather use something that's open.
- JavaFX needs to show that it works well first

# The 3x Future of Programming Languages

## 1. Java -- Structured, "high road"

- "**Programmer efficient**"
- GC for memory
- Great libs of off-the-shelf code
- Robust type system, anti-virus, anti-bug safety, checking of things at runtime (makes it a bit slower)
- Typed -- has a real type system to structure the code (makes it a bit verbose)
  - Most attractive for large and team projects
  - Attractive for APIs to be used by others (i.e. Eclipse auto complete)
- Speed is very good in some ways, but weak in some ways and startup time is not great. HotSpot paradigm of speedup via JVM optimization looks very good going forward.
- Memory use is high
- Now that Java is open source, Java is an extremely safe, open infrastructure technology if you just want things just to work.
- An excellent default choice unless there is a mismatch between the problem and java
- Open source projects tend to attract good community iteration and improvement (e.g. Hotspot research) -- we'll see how that plays out for Java.
- C# is like Java but locked in to Microsoft. Use it like Visual Basic -- to implement things that only need to work on Microsoft OS.

## Java Niche / Future

- Large, complex project -- play to Java's strengths in safety, structured type system
- Use very often for the server-side of Web applications
- Benefits from programmer efficiency (as Moore's law runs along, programmer efficiency looks better and better. Hardware gets cheaper, programmer time to create correct software does not!)

## Java Weaknesses

- Typing can be annoying
- Runs a little slower, users more memory
- Portable, but hard to access OS-specific features
- GUI can look ok, but hard to make it look really great

## 2. C/C++ -- "old road" / "fast road"

- Can run fast, use the minimum memory
- Not portable in the way that Java is, but can access system specific features
- The standard lib is tiny, but on the other hand, there are totally free, open source versions
- IMHO C/C++ is best for things that are not that complicated, and possibly where performance really matters.
  - e.g. disk driver, JPEG decompression -- having a nice, low-level C version to build on is great.
  - I wonder if the optimal system of the future will have some low-level parts in C, but with the upper, "system" part with a lot of custom coding in Java. From the Java side, you may not be aware what "native" C parts there are under the hood.
  - As a system, gets large, complex, and has a lot of people... it's tempting to build it in a safe system such as Java
- Legacy -- many things nowadays are in C/C++ because they were started 10 years ago and still work today (e.g. Acrobat, MS Word), or things where performance really matters. If Microsoft could snap its fingers and have MS Word in Python or C#, they would do it in a second. It's in C++ now because it was that way 10 years ago.

## 3. Python / Ruby / Javascript -- Scripting, "low road"

- Short little programs -- many projects have some part where this is a good fit (e.g. unit tests)
- Lacks type system in the code -- just declare variables/parameters and go -- very neat in its way

  - Makes the code short/dense, but hard to scale up to large projects
  - A language without declared types is great for 2 page programs. IMHO, typing is very useful for a 100,000 line project worked on by many people.
  - Contrast to Java large/team project
- Neat projects: "bean shell" and "Groovy"
  - Integrate scripting languages closely with Java -- run in JVM, access same classes objects
  - scripting languages integrated to work closely within Java -- use it for parts -- unit tests, allowing end-user scripting.

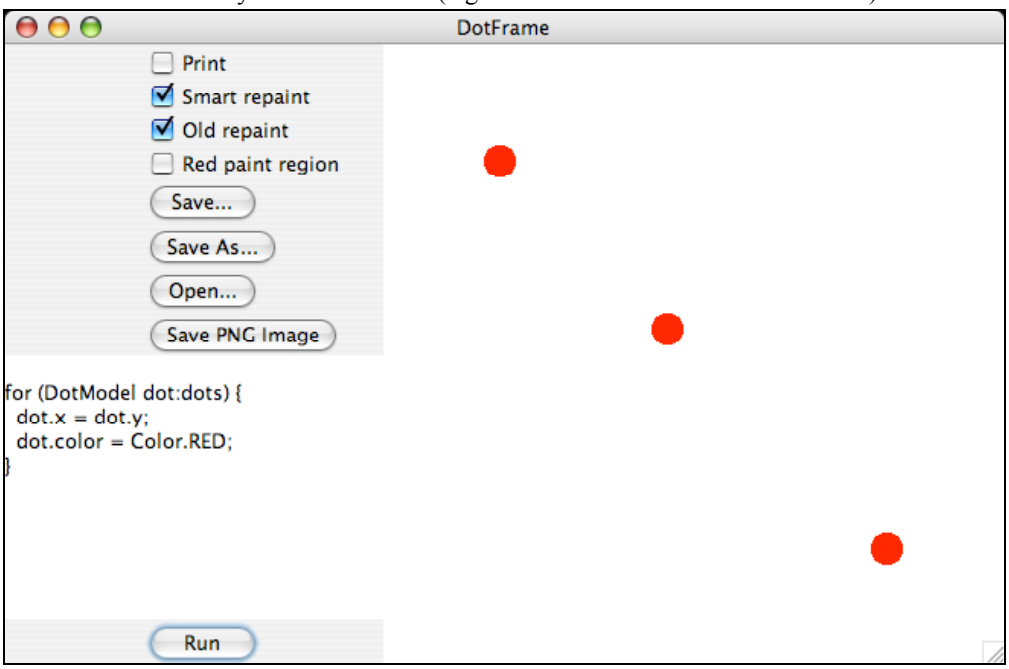
# Java Bytecode Ecosystem -- Hybrid System

## Java Ecosystem

- JVM/bytecode is a high quality, stable runtime environment -- especially now that Java is open source
- Other languages can be compiled to bytecode -- run in the JVM
- e.g. Jython project compiles Python to bytecode, JRuby runs ruby in the JVM

## BeanShell Example

- BeanShell is a very simple scripting language
- http://www.beanshell.org/
- BeanShell code is compiled on the fly to bytecode which interacts with full Java class and data structures
- Beanshell code understands Java objects, methods etc., integrating nicely with all that infrastructure
- Can have a Beanshell text area, the end user types some code in and hits run ... it compiles to bytecode and runs on the fly within the JVM. (e.g. the text area at the lower left below)




In this example the "dots" ArrayList has been exported to the Beanshell context at the lower left. The user can write Beanshell code there, hit the Run button, and the code is compiled and run against the JVM and all its objects.