

Embedded Software Development with Java?

By

Martin Astradsson

email: martin.astradsson@hyphen-innovation.com

Agenda

- Embedded SW Development with Java?
- Why are Java “not” used for embedded software?
- Why would it be beneficial to use Java?
- What are the problems using Java?
- Introduction to Java platforms, configurations and profiles
- Different places to start looking for information
- Conclusion

Why Are Java Not Used?

Business:

- Legacy code is in C/C++
- No time to learn a new language and tools
- Tied to a expensive vendor tool/library purchase
- Legacy hardware

Emotional:

- Want to stay in the C/C++ comfort zone
 - new way of developing, e.g. no pre-processor, pre-verification, no pointers, “no linking”

Technical:

- Java is too big
- Java is to slow
- Java is un-predictable (scheduling, garbage collection, dyn. class loading)
- Don't “trust” the JVM since I don't know what it is doing

Benefits

- Large standard libraries
- Easier to test applications on PC hosts without making wrapper layers
- RTOS independent application code
- Standard API's (networking, serial, parallel, bluetooth, usb,)
 - vendor independence, if needed your vendor change, but your application code does not.
- Java programs are more reliable (no memory leaks, array bounds checking)
 - and this also gives a lot of problems
- Java has exception handling (just like C++). Good,
 - but also a problem, normally people don't want to use C++ exceptions in real-time systems
- Primitive data type sizes are standardized (no more DWORD, int8, UINT32)
- Productivity / maintainability

Benefits

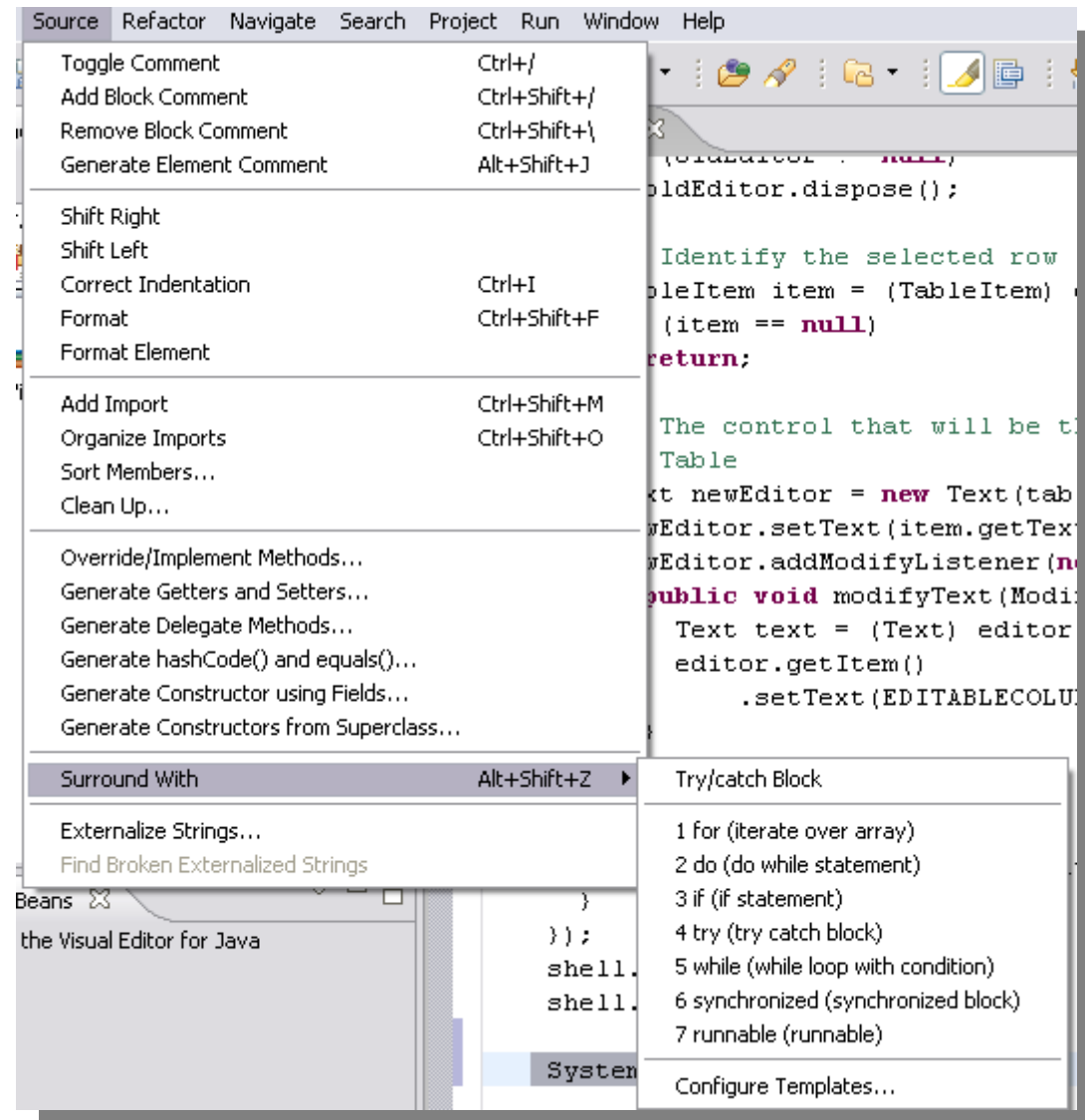
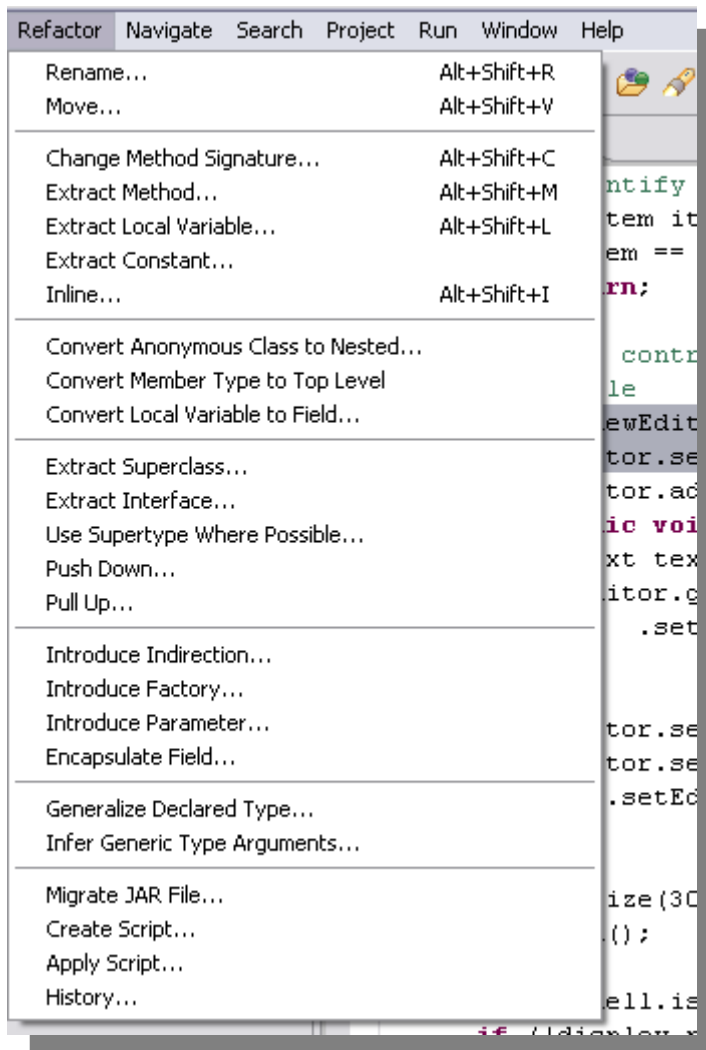
- Tools support are very good (Eclipse, NetBeans, Jedit, ..)
- JavaDoc and JUnit
- UML (reverse engineering, roundtrip, UML <-> Code sync via AST etc.)
- Source editing (quick fix, hyperlinks, java doc, api knowledge, etc..)
- Refactoring is really good
- Instant “problem” feedback when editing code
- Same tools and language from enterprise systems, over desktop, to large and small embedded systems with real-time requirements.
 - Team members can work in many areas
 - How many times have you been forced to learn new tools and compiler tool chains
 - How many RTOS' have you programmed for etc.
- Standard build systems (e.g. ANT) , continues integration etc.
 - Can you build you projects outside the vendor IDE?

Business Opportunities

- Productivity (e.g. refactoring, the code, compile immediately, document)
- Maintainability (JUnit, JDoc, soft/hard language)
- Mobile market:
 - 15 billion USD (downloadable application market by 2008)
 - cross 1 billion Java enabled units mid-2006
 - 30+ different vendors
 - 600+ models
- Make component oriented development
 - develop and test on PC
 - use them in embedded systems
 - use them on mobile devices (which is “embedded” in terms of memory constraints, debug capabilities)
 - use them for desktop applications

Productivity

Refactoring and Source Editing



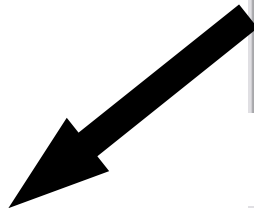
Maintainability

This is all you type, then press enter!

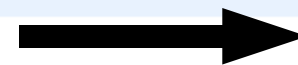
```
1
2 /**
3  * This is The Foo interface.
4  * @author astradss
5  *
6  */
7 public interface FooInterface
8 {
9     /**
10    public int foo(int a, int b);
11 }
12
```



```
Hello.java *FooInterface.java *Foo.java
1
2 /**
3  * This is The Foo interface.
4  * @author astradss
5  *
6  */
7 public interface FooInterface
8 {
9     /**
10    *
11    * @param a
12    * @param b
13    * @return
14    */
15    public int foo(int a, int b);
16 }
17
```



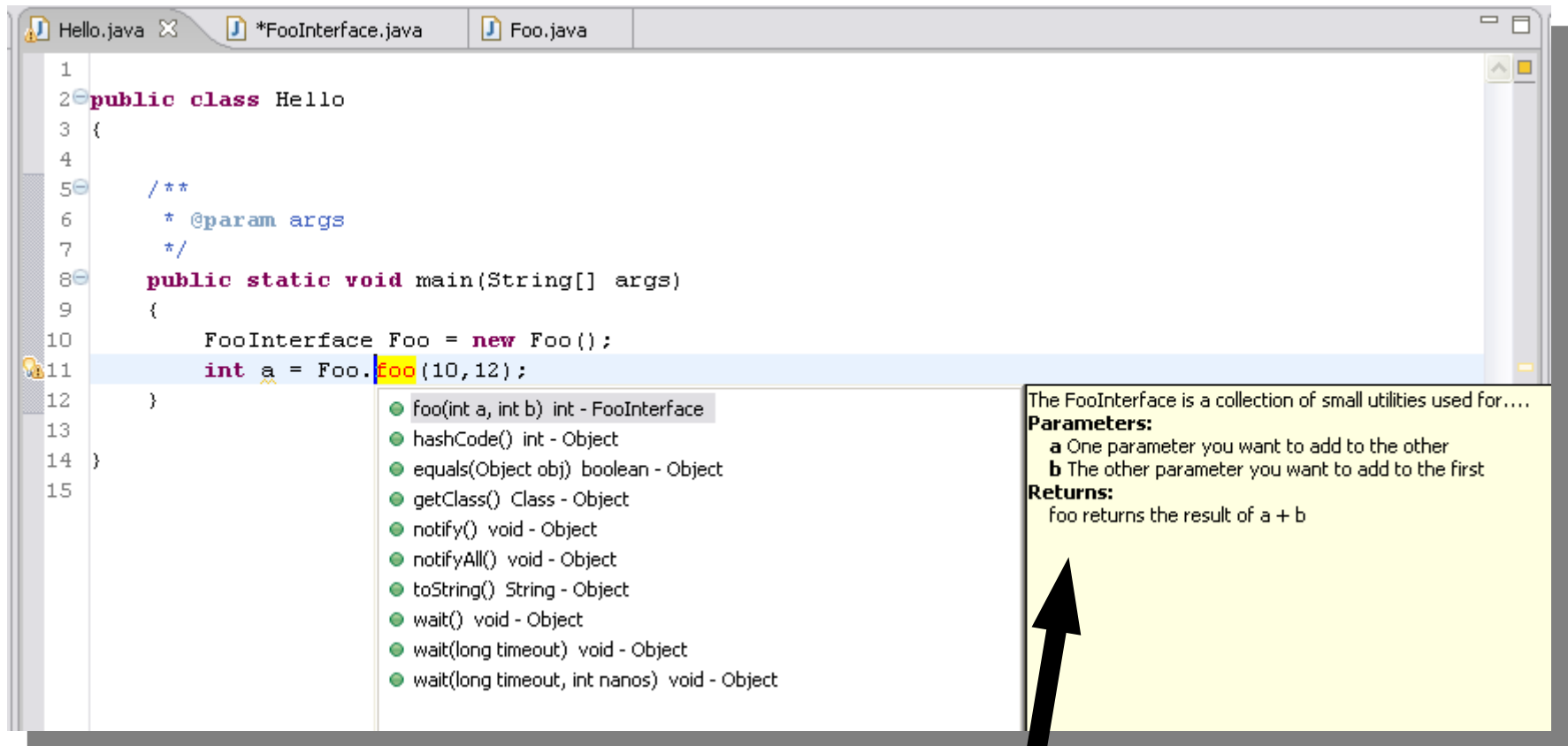
```
7 public interface FooInterface
8 {
9     /**
10    * The FooInterface is a collection of small utilities used for....
11    * @param a One parameter you want to add to the other
12    * @param b The other parameter you want to add to the first
13    * @return foo returns the result of a + b
14    */
15    public int foo(int a, int b);
16 }
17
```



```
Hello.java FooInterface.java *Foo.java
1
2 public class Foo implements FooInterface
3 {
4
5     public int foo(int a, int b)
6     {
7         return a+b;
8     }
9
```

Create the implementation

Maintainability



The screenshot shows an IDE window with three tabs: Hello.java, *FooInterface.java, and Foo.java. The main editor displays the following code:

```
1
2 public class Hello
3 {
4
5     /**
6      * @param args
7      */
8     public static void main(String[] args)
9     {
10        FooInterface Foo = new Foo();
11        int a = Foo.foo(10, 12);
12    }
13
14 }
15
```

A tooltip is visible over the `foo` method call on line 11. It lists the following methods:

- foo(int a, int b) int - FooInterface
- hashCode() int - Object
- equals(Object obj) boolean - Object
- getClass() Class - Object
- notify() void - Object
- notifyAll() void - Object
- toString() String - Object
- wait() void - Object
- wait(long timeout) void - Object
- wait(long timeout, int nanos) void - Object

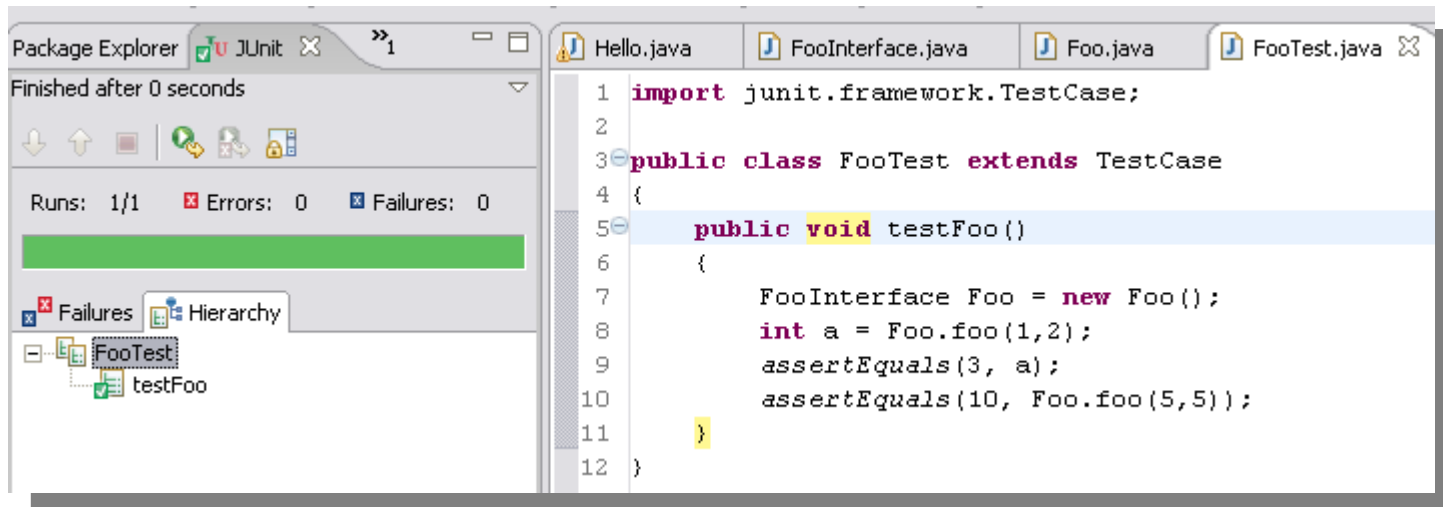
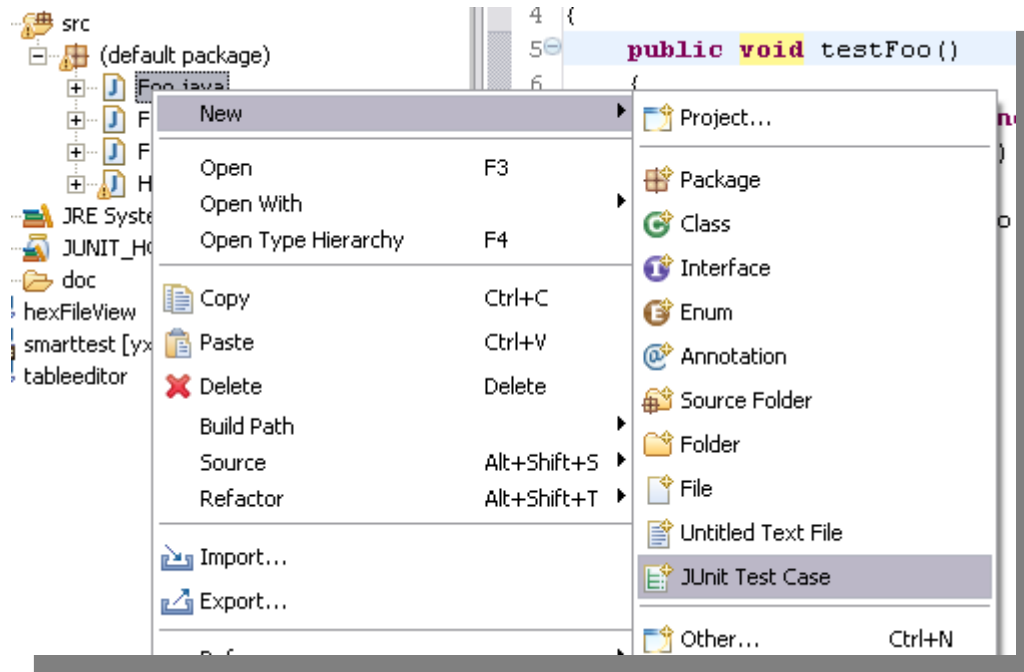
The tooltip also contains the following text:

The FooInterface is a collection of small utilities used for
Parameters:
a One parameter you want to add to the other
b The other parameter you want to add to the first
Returns:
foo returns the result of a + b

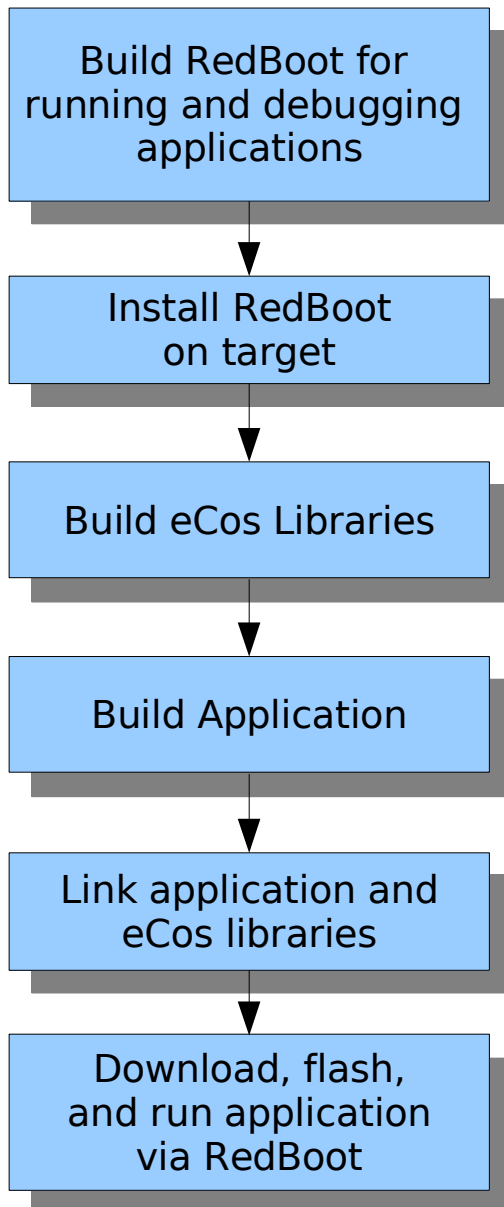
An arrow points from the text box below to the tooltip.

You own API's is easily accessible within the tool for the other developers in the projects.

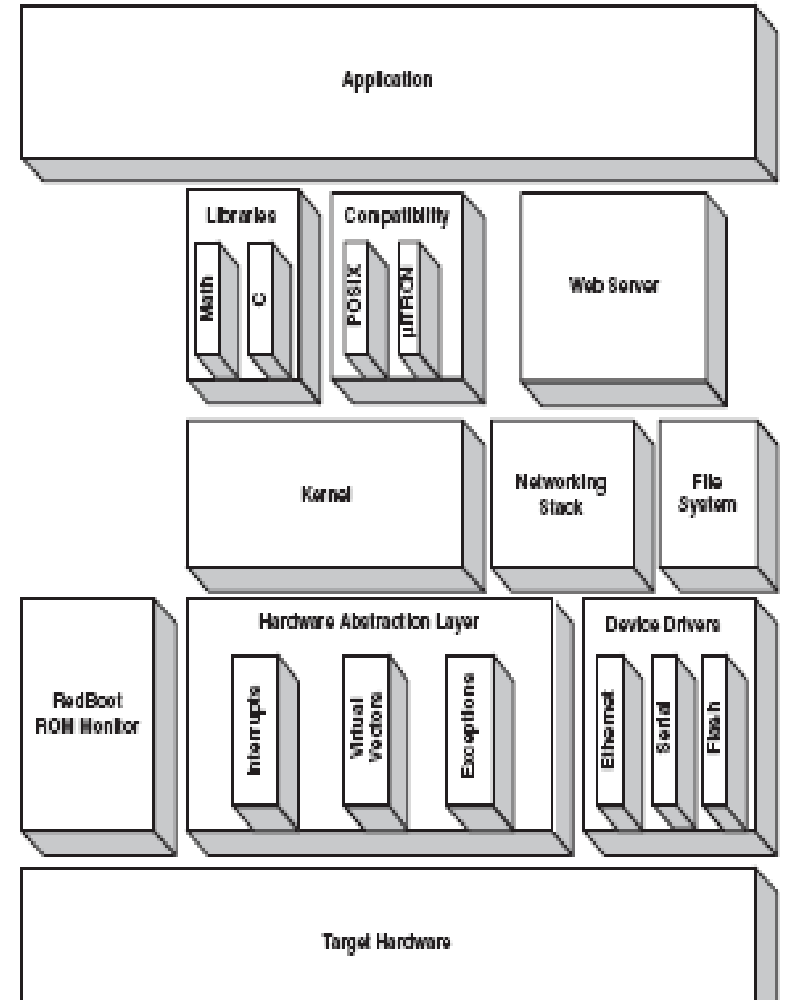
Maintainability



C/C++ Development Flow

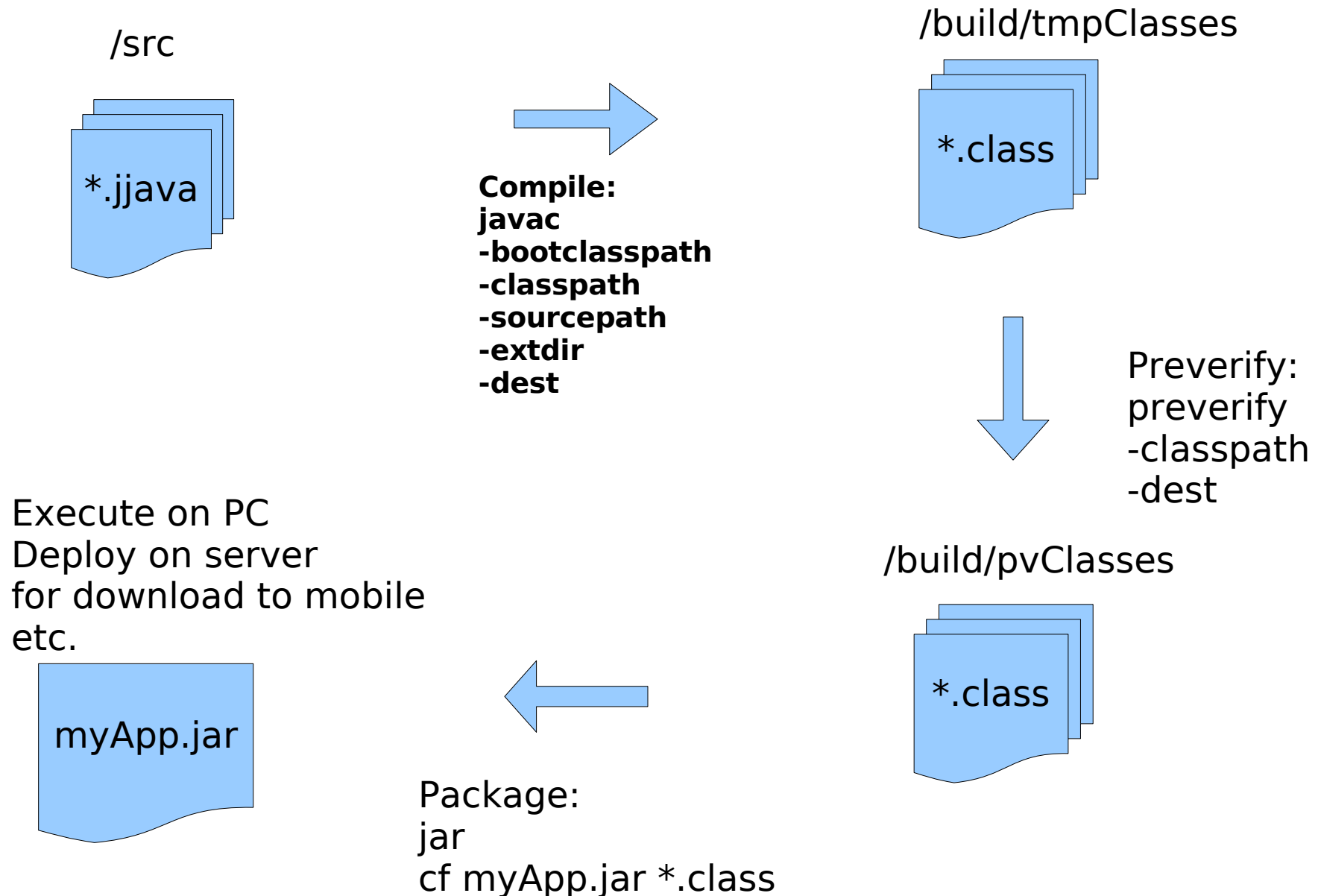


Configure source tree, then build



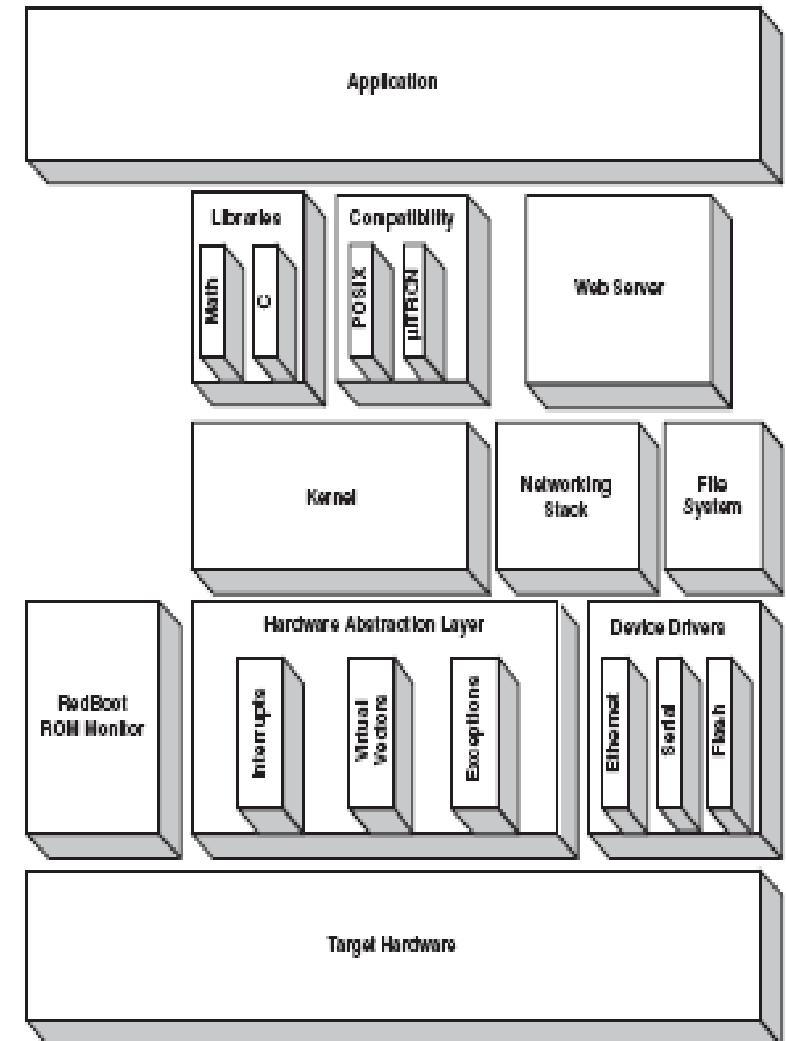
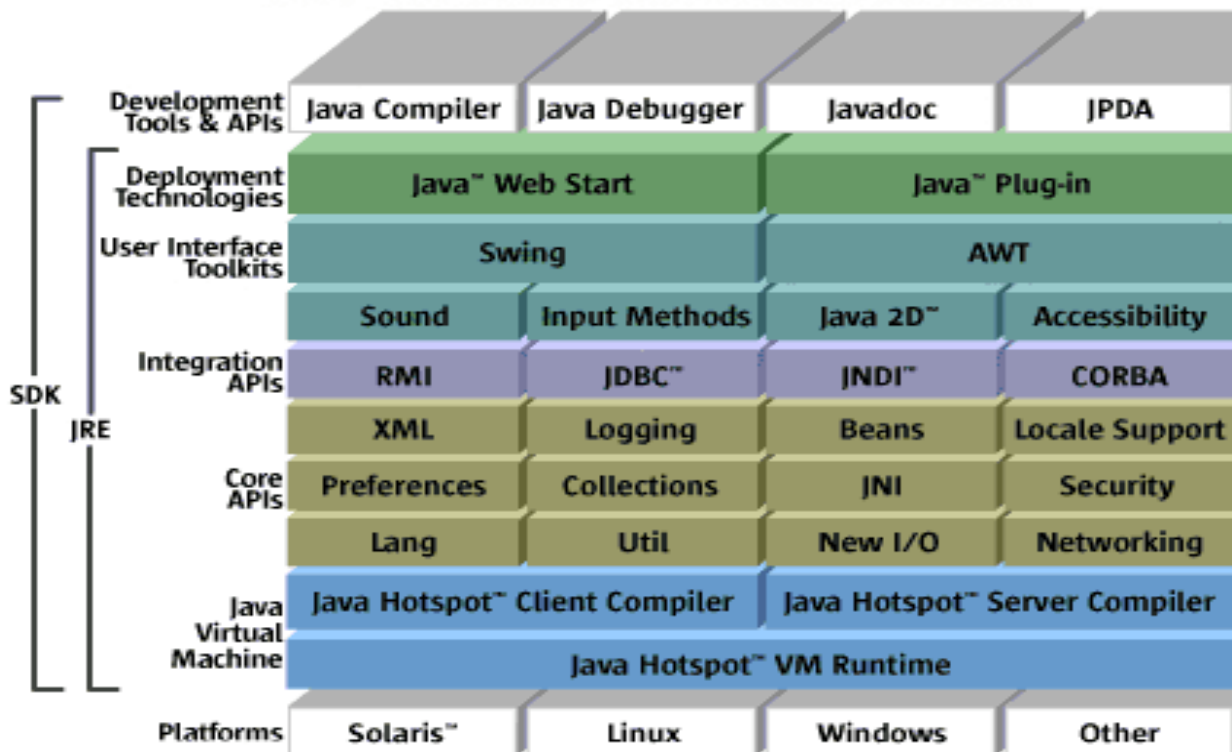
You know what is happening all the way from HW reset, boot code, boot loader, interrupts, until your framework is started etc. You are in control with respect to memory layout via linker command files and pragma directives

Java Development Flow



Put People Back in the Comfort Zone

Java™ 2 Platform, Standard Edition v 1.4



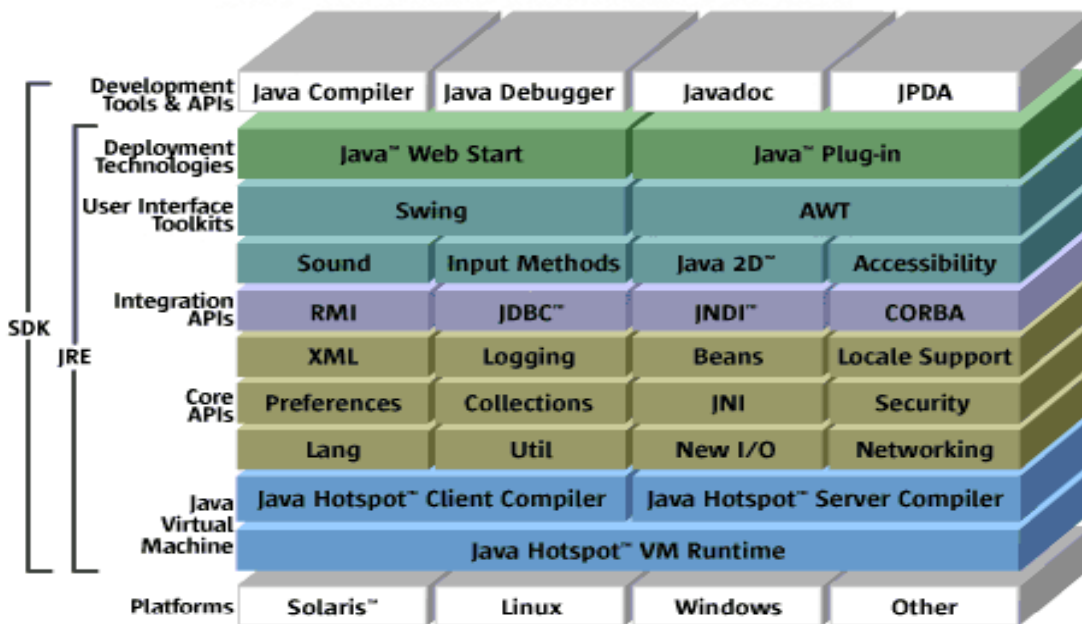
Java Platforms

Some embedded devices are maybe so big and complex today that they could use a standard platform. (maybe with the RTSJ extension which is available from Sun)

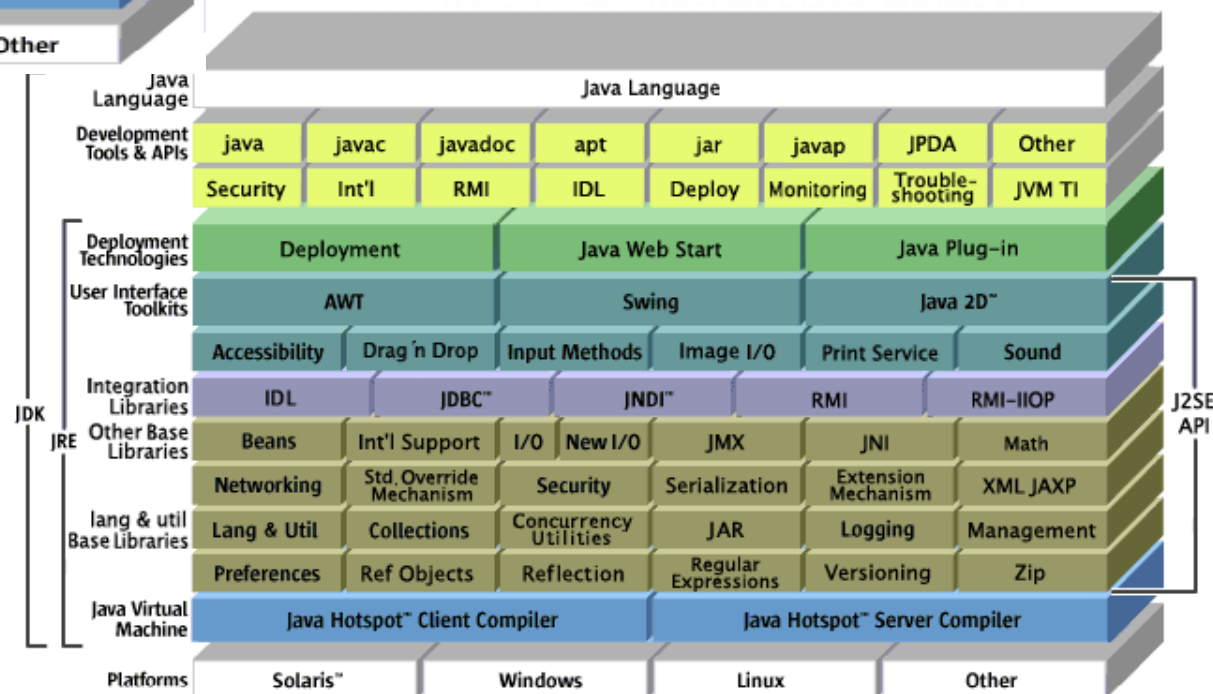
If you can choose go for 5.0.

- Concurrency support has improved
- With state machines, you will welcome enums
- and of course generics

Java™ 2 Platform, Standard Edition v 1.4



Java™ 2 Platform Standard Edition 5.0



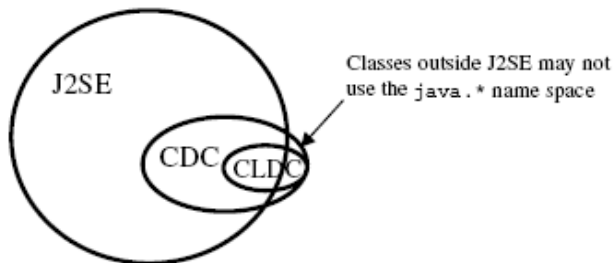
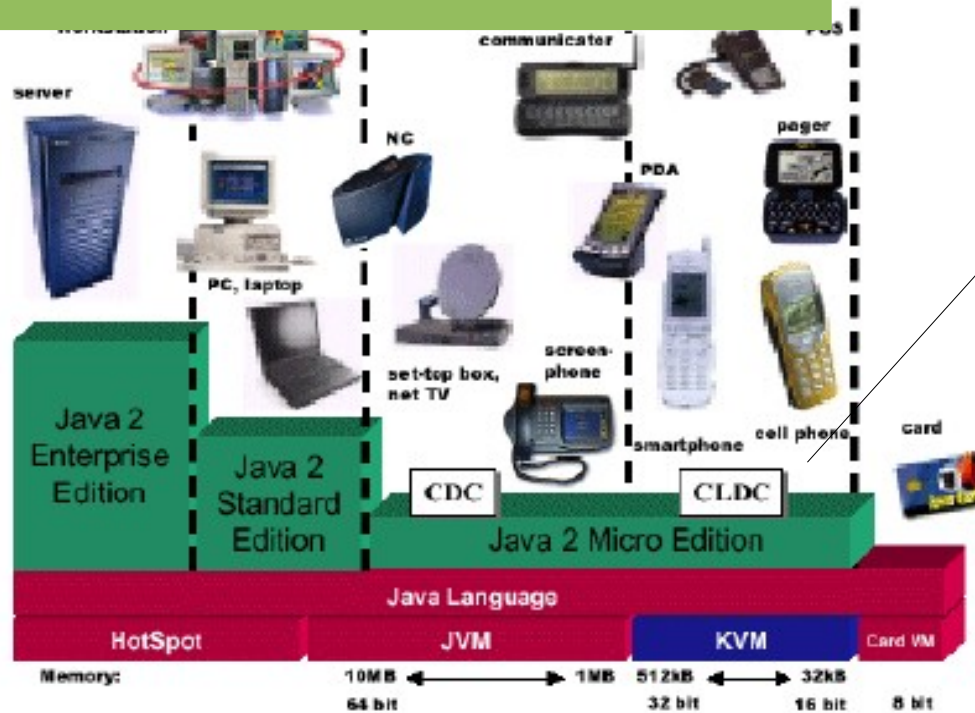
Editions, Configs and Profiles

Not included in CLDC configuration:

- Application life-cycle management (application installation, launching, deletion)
- User interface functionality
- Event handling
- High-level application model (the interaction between the user and the application)

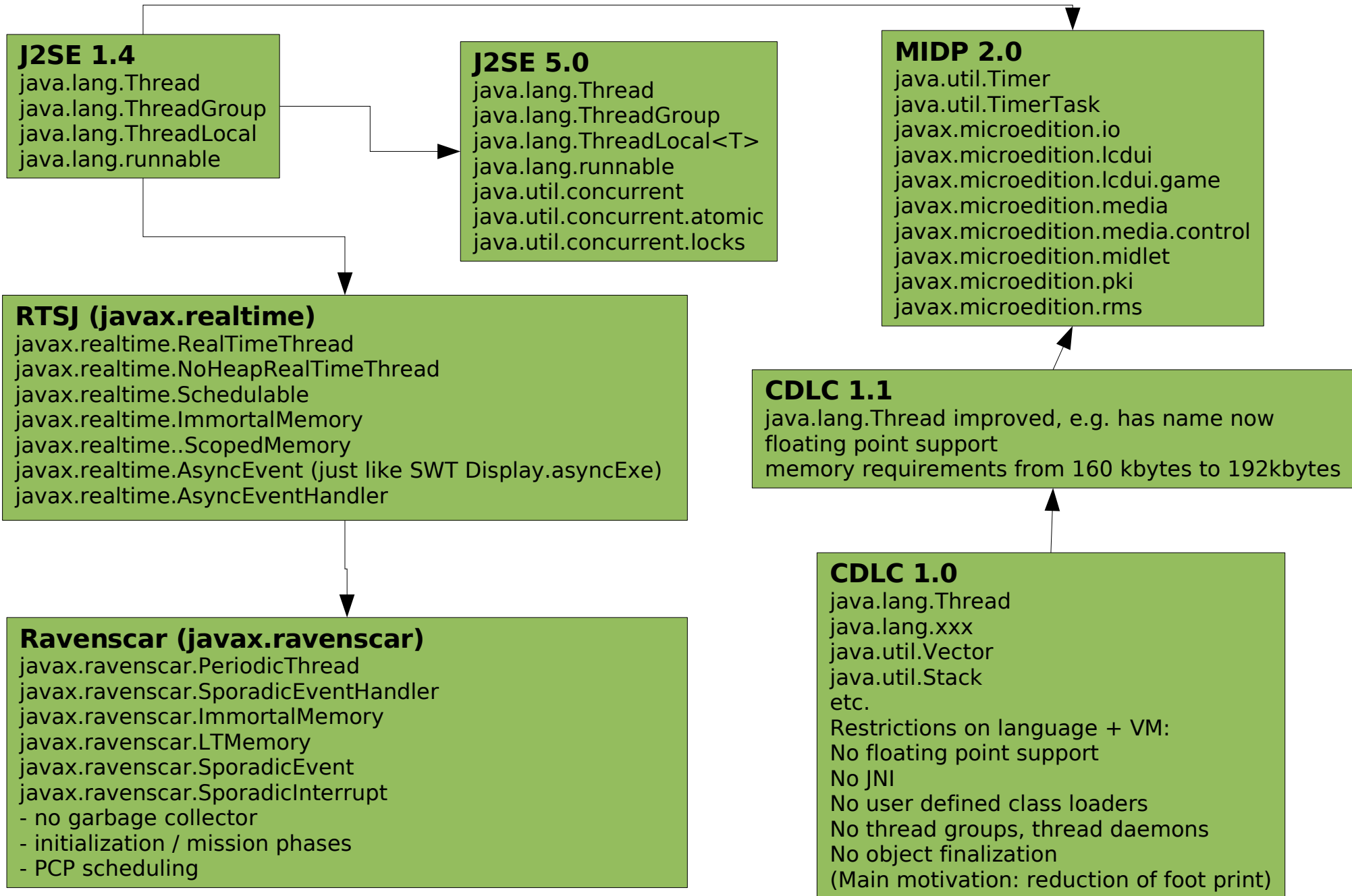
Configuration defines:

- set of java language features
- minimum VM features
- a set of class libraries



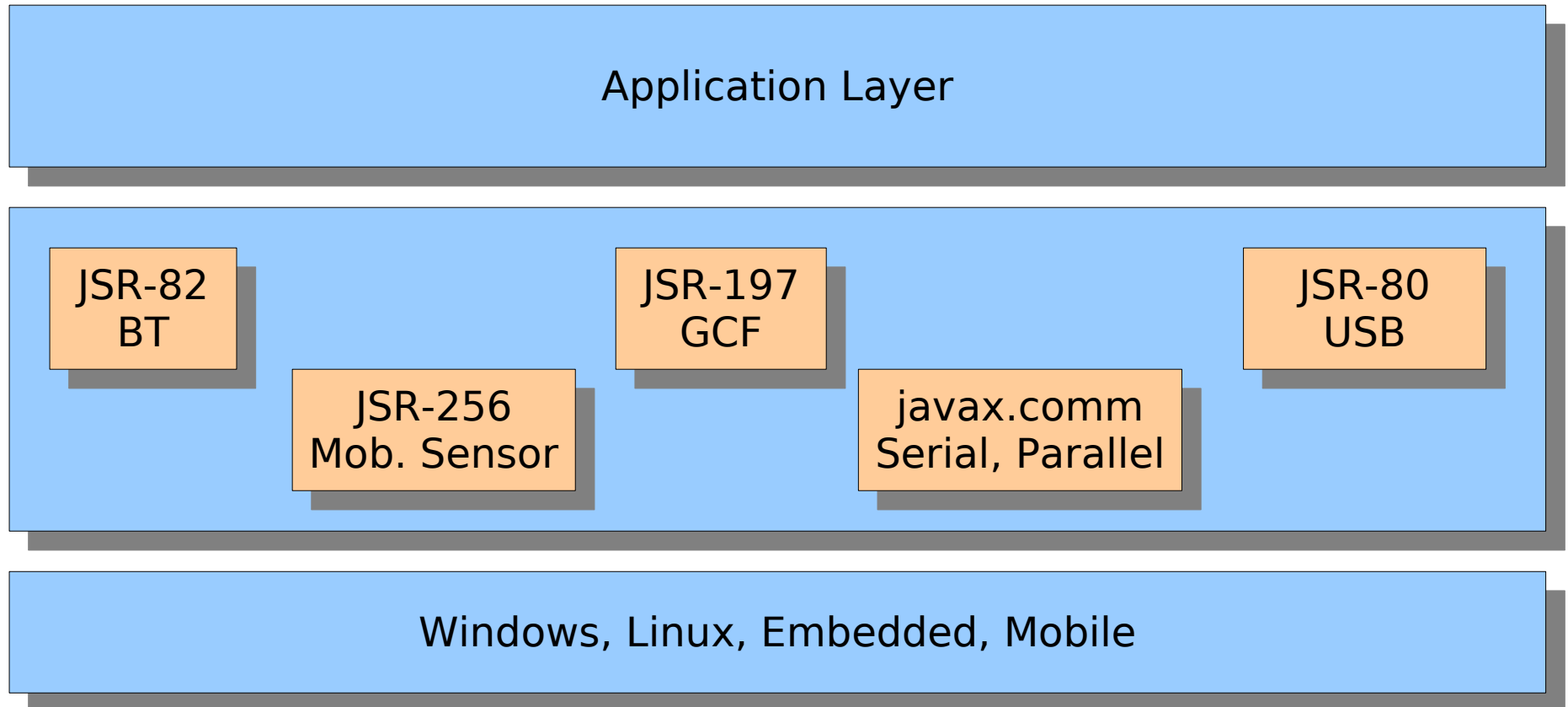
- The goal of the configuration is to guarantee portability and interoperability between various kinds of resource-constrained devices
- Therefore the configuration shall not define any optional features.
- This limitation has a significant impact on what can be included in the configuration and what should not be included.
- The more domain-specific functionality must be defined in profiles rather than in CLDC.

Editions, Configs and Profiles



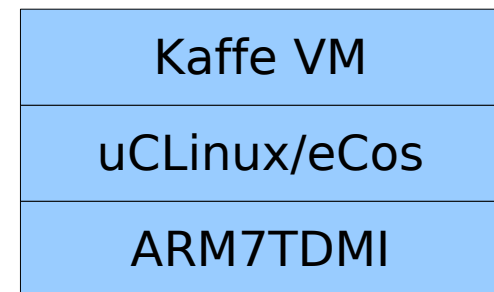
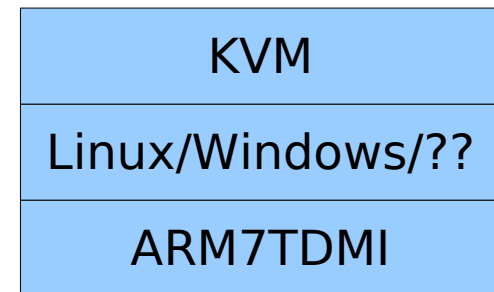
Standard API's

- These can help you as an embedded developer



Want to Use it in Next Project Software Solutions

- Sun KVM
 - 128kbyte for the VM and its libraries
- Sun HotSpot
 - 512kbyte to 1Mbyte for the VM stack
 - Commercial license
- Kaffe (www.kaffe.org)
 - Linux dist*, uCLinux
 - Windows, Windows CE. DOS
 - ThreadX, eCos, VxWorks, RTEMS
 - Processors, e.g. ARM, x86
- GCJ (<http://gcc.gnu.org/java/>) + jRate (RTSJ add on to GCJ)
 - Also ARM7 with Newlib



Want to Use it in Next Project

Byte codes and the VM

- Jamaica virtual machine Macro Assembler

- an assembly language for JVM byte code programming
- generates java class files

- ASM a byte code manipulation framework

- use it to see what byte code would generate what java code
- or what does the byte code for this java code looks like?
- Has also an Eclipse plug in

```
public class CFirstCls  
{  
    int count;
```

```
    public CFirstCls() {  
        iconst_0  
        putfield count int  
        return  
    }
```

```
    public void inc(int amount) {  
        getfield count int  
        iload amount  
        iadd  
        putfield count int  
        return  
    }
```

```
}
```

- Jakarta BCEL

- Byte Code Engineering Library
- Use it to investigate and understand java class files.

- Jasmin

- Yet another Java assembler

Want to Use it in Next Project Hardware Solutions

- Ajile systems (www.ajile.com)
 - J-100, 99% of byte codes in HW
 - No RTOS
 - 48kbyte internal RAM, no Flash :-)
- GUI based application build configuration and control tool - JEM Builder
- Utilizes standard JVM class files generated by commercial Java IDEs
- Statically resolves class files and eliminates unused methods and fields
- Performs byte code optimizations
- Builds boot tables, class initialization code, and assigns interrupt and trap handlers
- Configures JVM's and memory layout
- Price tag?



Want to Use it in Next Project Hardware Solutions

- Systronix
 - Many different HW boards / solutions
 - JStick, JStamp
 - They have a comparison matrix <http://jstik.systronix.com/compare.htm>
- Java Optimized Processor
 - FPGA solution
 - Well... we have already had the best possible introduction to that :-)

The Future

- Sun SPOT (Wireless Sensor Platform)
 - Small Programmable Object Technology
- VM's for small wireless controller devices and wireless sensor networks
- ARM7 CPU, flash memory and SRAM, as well as an 802.15.4 Zigbee wireless radio chip
- The VM is "Squawk"
 - Optimized J2ME VM written almost entirely in Java
 - Run on bare metal => no need for an RTOS
 - Pre-verifies as usual; all symbolic references are resolve etc.
 - Can run multiple suites
 - Can run multiple suites and each suite's class files is optimized for that
 - Prototype VM is 350 kbytes of RAM and Flash combined
- But, look here for the complete text with critiques:
- <http://www.embedded.com/showArticle.jhtml;jsessionid=LQLP2GZKQH0LWQSNL0SKHSCJUNN2JVN?articleID=188101293>



Conclusion

- Before starting out with an embedded Java project there are a great deal of information that need to consumed
- It is not enough to understand the Java language
- The JVM/KVM should be understood together with the boot process etc., just as the C runtime system is understood.
- Lot's of profiles and configurations should be studied
- Lot's of combinations of hardware and software solutions should be studied
 - But all this is business as usual when looking into pros and cons of new technology
- If the Embedded Java development environment where described better by vendors, as seen from a C/C++ programmers perspective, it's likely that more people would take the step
- We get higher productivity, better tools, better vendor independence and so on.

Embedded Software Development with Java?

By
Martin Astradsson
email: martin.astradsson@hyphen-innovation.com

Agenda

- Embedded SW Development with Java?
- Why are Java “not” used for embedded software?
- Why would it be beneficial to use Java?
- What are the problems using Java?
- Introduction to Java platforms, configurations and profiles
- Different places to start looking for information
- Conclusion

Why Are Java Not Used?

Business:

- Legacy code is in C/C++
- No time to learn a new language and tools
- Tied to a expensive vendor tool/library purchase
- Legacy hardware

Emotional:

- Want to stay in the C/C++ comfort zone
 - new way of developing, e.g. no pre-processor, pre-verification, no pointers, “no linking”

Technical:

- Java is too big
- Java is to slow
- Java is un-predictable (scheduling, garbage collection, dyn. class loading)
- Don't “trust” the JVM since I don't know what it is doing

3

There can be many reasons for developers not using Java for embedded software development. In an organization there can be political reasons, business reasons, emotional reasons and believes from individuals that has a strong say within the organization, and finally pure technical arguments that indicates that Java is not the right choice for a certain application.

All these area has to be addressed if one would like to introduce Java as part of the embedded software development strategy in a company.

Apart from the technical concerns the challenges that one has to overcome is not different from introducing any other change in a technical strategy within organizations.

Benefits

- Large standard libraries
- Easier to test applications on PC hosts without making wrapper layers
- RTOS independent application code
- Standard API's (networking, serial, parallel, bluetooth, usb,)
 - vendor independence, if needed your vendor change, but your application code does not.
- Java programs are more reliable (no memory leaks, array bounds checking)
 - and this also gives a lot of problems
- Java has exception handling (just like C++). Good,
 - but also a problem, normally people don't want to use C++ exceptions in real-time systems
- Primitive data type sizes are standardized (no more DWORD, int8, UINT32)
- Productivity / maintainability

4

There are many benefits when using Java and some of them are listed here. There is no doubt that if you have tried Java and good Java development tools, you will feel like you are missing something going back to the C/C++ world.

Some of the standard problems that an embedded software developer face is solved or partly solved with Java like data type ranges are standardized, you need only to learn one set of thread semantics not 2,3 or 4 different RTOS'es.

There are large standard libraries which again protect your investment in learning new API's, but of course you then have to navigate in the profile jungle instead, since some libraries are only available in standard edition and so on.

The hardware access is also supported by a lot of effort in standardizing API's.

Benefits

- Tools support are very good (Eclipse, NetBeans, Jedit, ..)
- JavaDoc and JUnit
- UML (reverse engineering, roundtrip, UML <-> Code sync via AST etc.)
- Source editing (quick fix, hyperlinks, java doc, api knowledge, etc..)
- Refactoring is really good
- Instant “problem” feedback when editing code
- Same tools and language from enterprise systems, over desktop, to large and small embedded systems with real-time requirements.
 - Team members can work in many areas
 - How many times have you been forced to learn new tools and compiler tool chains
 - How many RTOS' have you programmed for etc.
- Standard build systems (e.g. ANT) , continuous integration etc.
 - Can you build you projects outside the vendor IDE?

5

The tools for Java development is really good and even if the language didn't give any advantage compared to e.g. C++, the features of the tools will result in a better productivity.

JavaDoc and unit testing is better integrated and even you own documentation becomes an integrated part of the tool and makes it easier for fellow developers to use your code.

There are many UML tools available for Java and they support all from simple forward engineering to reverse engineering and full round trip. Even free versions of the tools are very good. The same level of C++ support is not seen since the parsing of C++ is not trivial.

You simply start thinking like a “better” software developer and start using java doc, unit test, continuous integration and so on, which again will improve your code base, productivity and quality.

Another benefit is that you could use the same language and semantic all the way from enterprise systems, over desktop down to embedded systems. Even web development including GUI could be developed with Java if for instance Google's web toolkit is used.

Business Opportunities

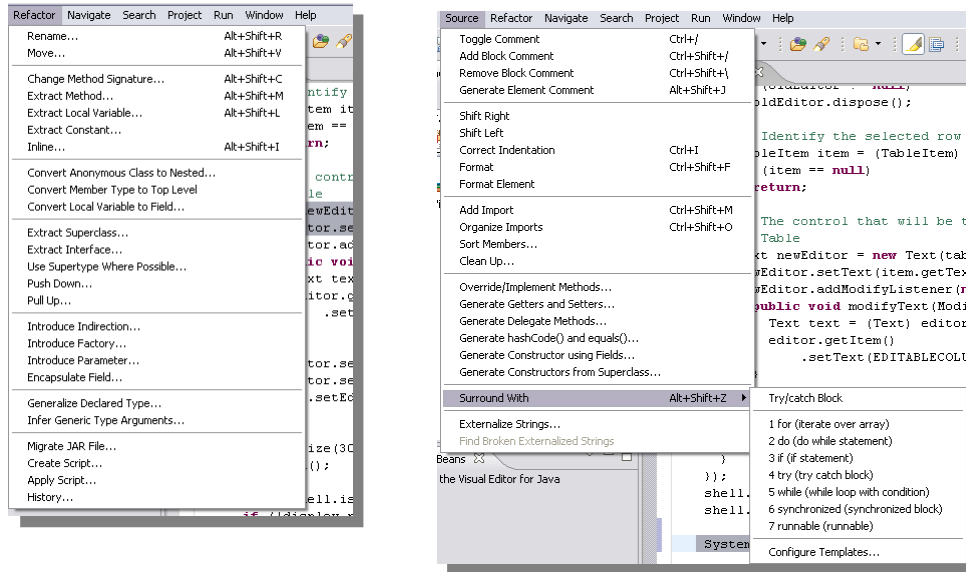
- Productivity (e.g. refactoring, the code, compile immediately, document)
- Maintainability (JUnit, JDoc, soft/hard language)
- Mobile market:
 - 15 billion USD (downloadable application market by 2008)
 - cross 1 billion Java enabled units mid-2006
 - 30+ different vendors
 - 600+ models
- Make component oriented development
 - develop and test on PC
 - use them in embedded systems
 - use them on mobile devices (which is “embedded” in terms of memory constraints, debug capabilities)
 - use them for desktop applications

6

Beside from productivity and maintainability, which are very important, one could also benefit from making components in Java and then use them on PC's, in normal embedded targets and on mobile phones. There is a high demand for wireless solutions also for instrument control applications and why not use some of the same business components on the desktop as on the mobile devices. Again the differences between profiles has to be taking into account from the beginning to make this possible.

Productivity

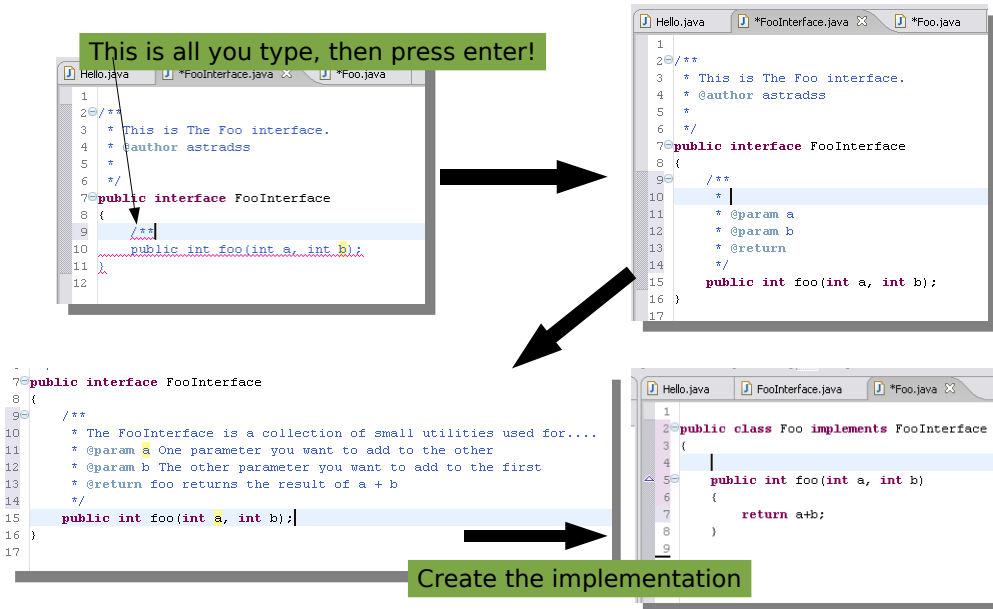
Refactoring and Source Editing



A example of the level of support for refactoring and source code changes in eclipse JDT. This has a great impact on the productivity and the improvements you do on the code base during development, since it it not a problem or major task to do it.

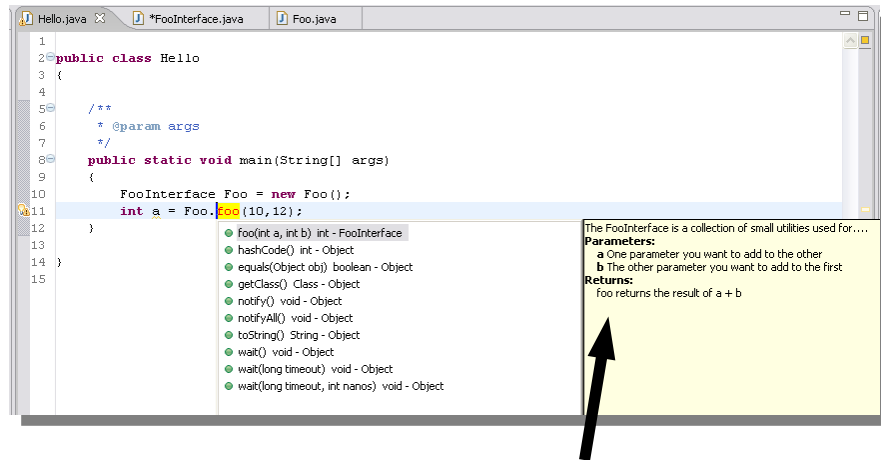
If you compare this to the refactoring supported by C++ IDE's (event the CDT, which are the C++ tools for eclipse) the number of refactoring s are only a few. And in some expensive tools from various vendors there are no support what so ever.

Maintainability



A small simple example of using java doc. If you have the method the Java doc tags are created automatically, so even for people who can not remember them there is not excuse not to document the code.

Maintainability

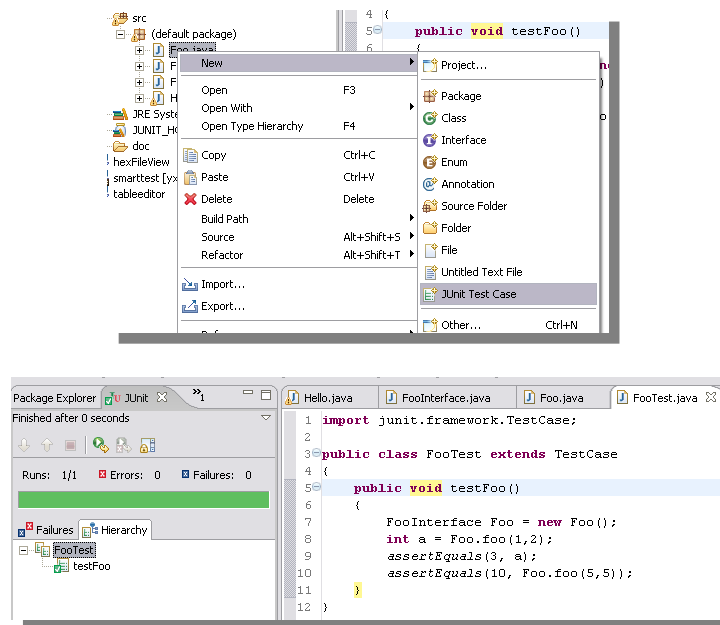


You own API's is easily accessible within the tool for the other developers in the projects.

9

The documentation is easily accessible and integrates well with the tool, so other developers can use your libraries (or your self for that matter).

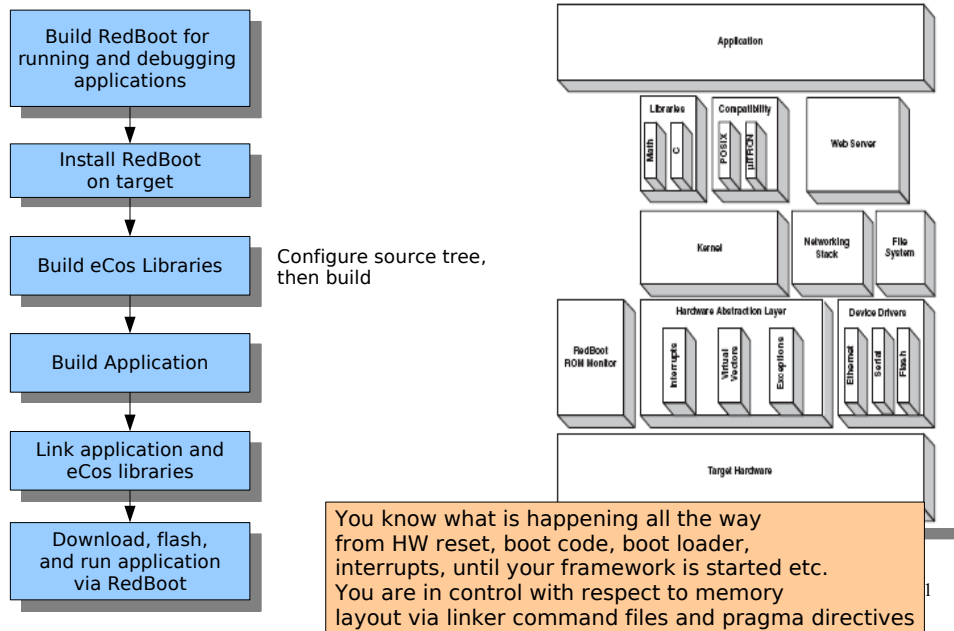
Maintainability



10

An example of the unit test integration in eclipse. It just makes is more simple and the net result is that you actually start to use it, which again will give you a better confidence in the code base, and will improve the quality over time.

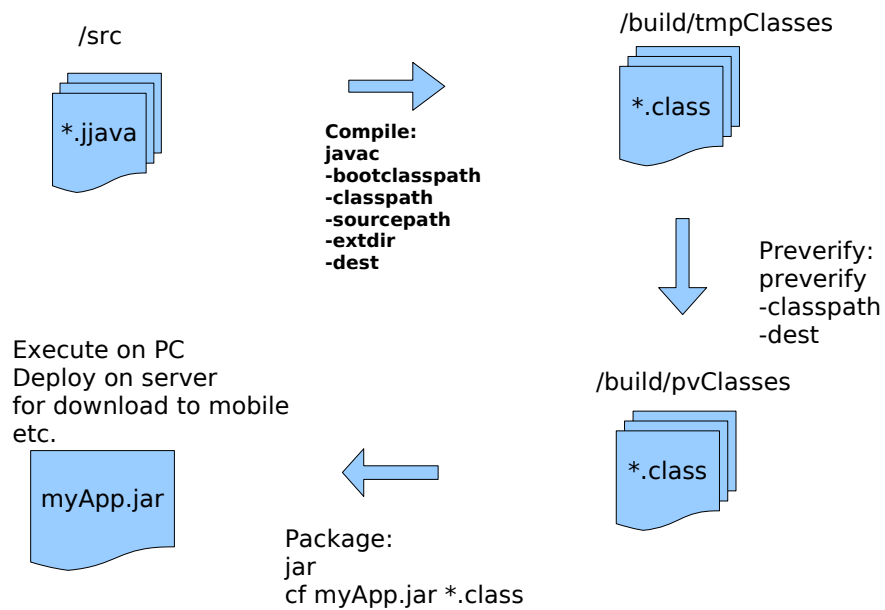
C/C++ Development Flow



Starting to use Java is also a shift in development process in some sense. In C++ (or C) embedded developers are used to controlling source code compilation with "ifdefs", they know how the system starts up all the way from the boot code, over the RTOS to the main of the framework or the application. They are used to write linker command files to place components in the expected memory areas (e.g. rtos in internal zero wait state memory), they know what goes into .bss, .text, .data and so on, and also know how the C runtime system initialize memory segments. The linking is done using object files or libraries and the final image is downloaded to the target as a binary image, a hex file or even as an elf file. In the target it is downloaded to ram and executed or flashed. When you link you need to consider if the image is going to be executed from the flash (load address) or from another ram runtime address, and so on.

It is all a well known scenery for embedded software developers. When shifting to Java this landscape is disturbed and you take the developers out of their comfort zone. They don't know what is happening any more from the point of pressing power on until main is called.

Java Development Flow

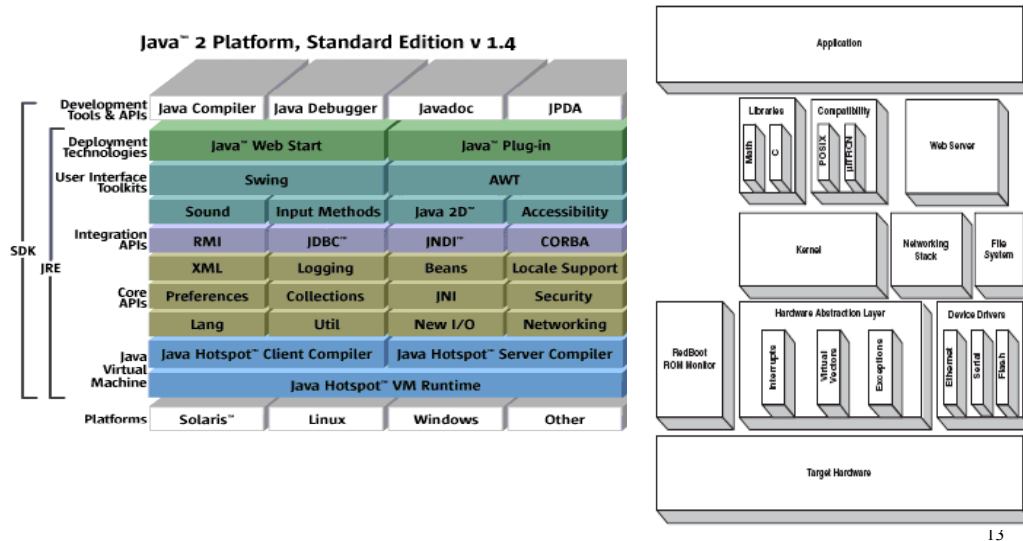


12

The development flow is changed. The most obvious one is the lack of a preprocessor, and for embedded deployment you also have the preverification step.

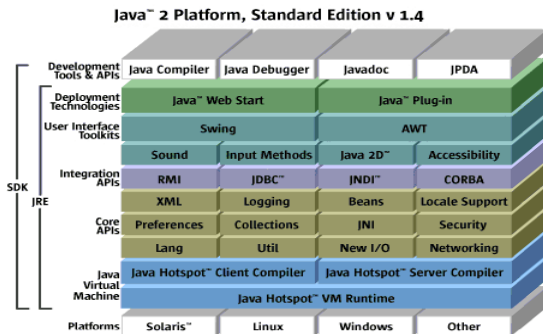
Normally on the desktop dynamic class loading, user defined class loaders and so on are part of the landscape, but for embedded deployment you don't have these options and you can not call the jvm with the jar file for starting the application.

Put People Back in the Comfort Zone



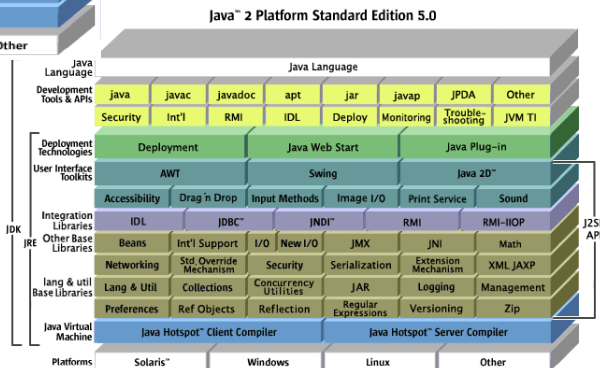
So in order for new developers to stay in their comfort zone and being willing to take the step and start using Java for embedded development, they have to understand the normal C/C++ development steps in terms of Java semantics and development flow and it involves both tools, libraries and applications, and differences in linking (and also the desktop Java approach versus the embedded Java approach), jvm's, kvm's all the profiles and so on.

Java Platforms



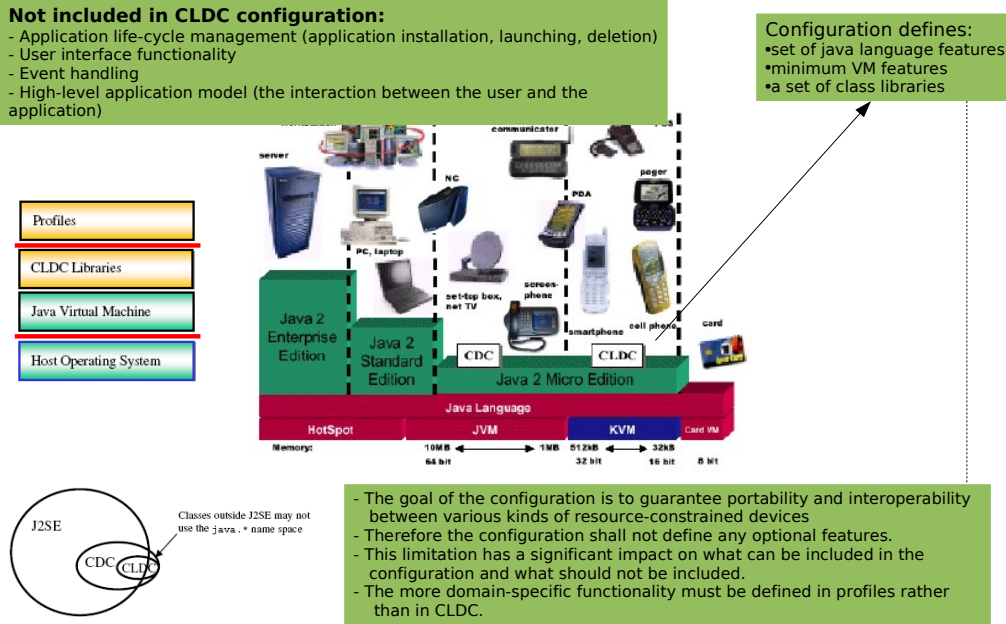
Some embedded devices are maybe so big and complex today that they could use a standard platform. (maybe with the RTSJ extension which is available from Sun)

- If you can choose go for 5.0.
- Concurrency support has improved
 - With state machines, you will welcome enums
 - and of course generics



The first thing that someone new to Java face is the the different platforms for enterprise development, over desktop to embedded in terms of the micro edition or down to java card. Each of these also has its own life cycle with different set of features and libraries, e.g. the introduction of enums and generics in Java 5.0 which certainly is a big advantage, but also the improved support for concurrent application development.

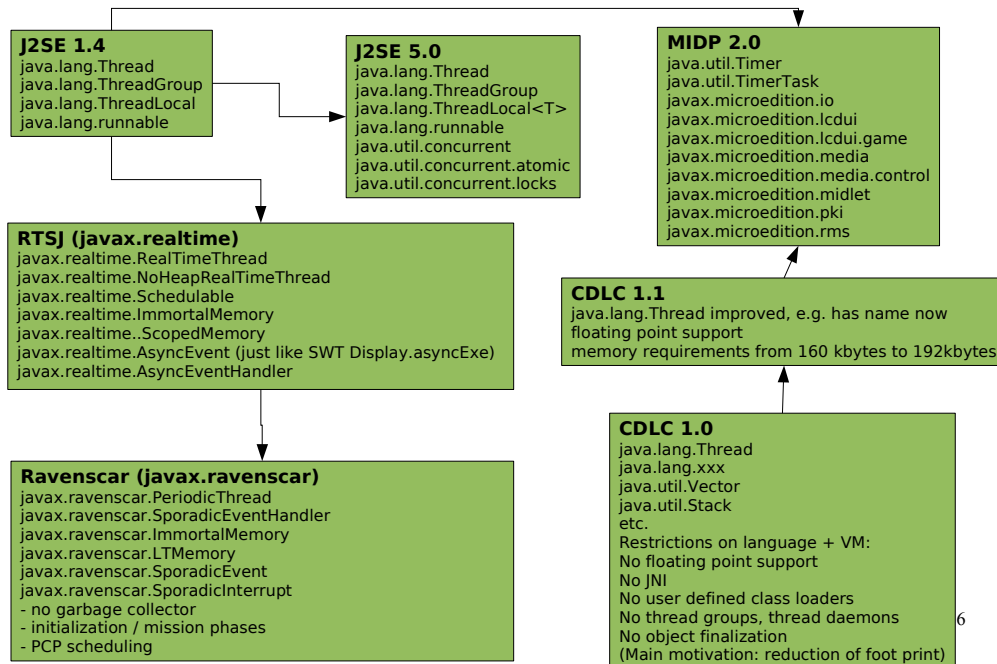
Editions, Configs and Profiles



When you are new to the java world all the different editions and platforms and configurations – which exists in different versions like Java Card 2.0, or 3.0, J2SE 1.4.x or J2SE 5.0, J2ME with CLDC 1.0 or 1.1, and so on. And on top of this comes profiles like the MIDP 1.0, MIDP 2.0, Java Ravenscar Profile, Real time specification for Java...

This can be some confusing in the beginning and it is important to understand the differences in API's, libraries and JVM features if one would like to develop components that can be used in more than one setup.

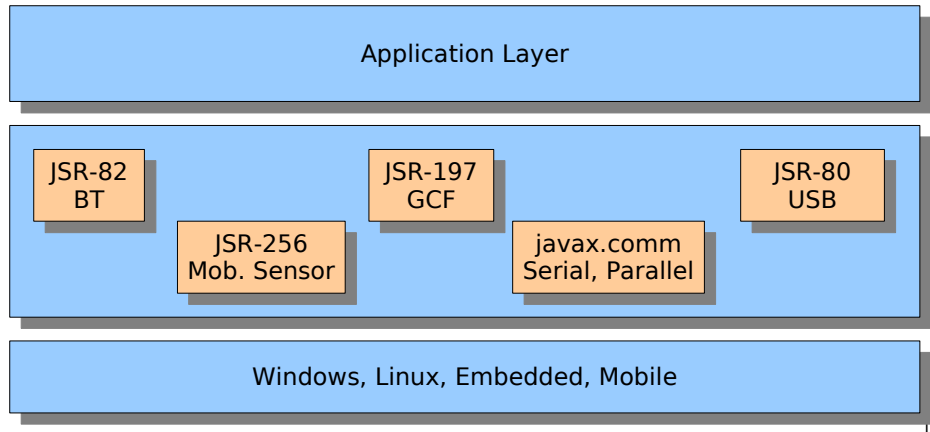
Editions, Configs and Profiles



Just a simple example showing some differences between editions, profiles and configurations. It is very simplified but should give a idea about how things evolves or depends on each other.

Standard API's

- These can help you as an embedded developer



In embedded software development it is normal to make hardware abstraction layers to make application (business) components independent of the hardware platform. In many cases companies decide to purchase vendor components and then write the business logic against the vendor API. The problem with this scenario is that the API is not standardized and if not wrapped with an in-house API the component could not be changed with a better component without rewriting part of the business components.

If standardized APIs are used (and requested from vendors) the vendor lock-in problem is avoided. If you find a better (faster/cheaper/..) component you can switch it without changing your business logic. You can even use an JSR-82 bluetooth API on the PC for easier testing and then deploy to target for final testing later in the development process (or just use the same BT application on the desktop and mobile). The same could be true for using serial communication. Most micro controllers have UARTs and most people wrap them in an in-house API. The javax.comm could be used and then the application could be tested on the PC. There are a lot of interesting APIs on the Java side, and these could over time make the embedded applications more platform independent.

Want to Use it in Next Project Software Solutions

- Sun KVM
 - 128kbyte for the VM and its libraries
- Sun HotSpot
 - 512kbyte to 1Mbyte for the VM stack
 - Commercial license

KVM
Linux/Windows/??
ARM7TDMI

- Kaffe (www.kaffe.org)
 - Linux dist*, uCLinux
 - Windows, Windows CE, DOS
 - ThreadX, eCos, VxWorks, RTEMS
 - Processors, e.g. ARM, x86

Kaffe VM
uCLinux/eCos
ARM7TDMI

- GCJ (<http://gcc.gnu.org/java/>) + jRate (RTSJ add on to GCJ)
 - Also ARM7 with Newlib

Want to Use it in Next Project

Byte codes and the VM

- Jamaica virtual machine Macro Assembler
 - an assembly language for JVM byte code programming
 - generates java class files
- ASM a byte code manipulation framework
 - use it to see what byte code would generate what java code
 - or what does the byte code for this java code look like?
 - Has also an Eclipse plug in
- Jakarta BCEL
 - Byte Code Engineering Library
 - Use it to investigate and understand java class files.
- Jasmin
 - Yet another Java assembler

```
public class CFirstCls
{
    int count;

    public CFirstCls() {
        iconst_0
        putfield count int
        return
    }

    public void inc(int amount) {
        getfield count int
        iload amount
        iadd
        putfield count int
        return
    }
}
```

Want to Use it in Next Project Hardware Solutions

- Ajile systems (www.ajile.com)
 - J-100, 99% of byte codes in HW
 - No RTOS
 - 48kbyte internal RAM, no Flash :-)
- GUI based application build configuration and control tool - JEM Builder
- Utilizes standard JVM class files generated by commercial Java IDEs
- Statically resolves class files and eliminates unused methods and fields
- Performs byte code optimizations
- Builds boot tables, class initialization code, and assigns interrupt and trap handlers
- Configures JVM's and memory layout
- Price tag?



Want to Use it in Next Project Hardware Solutions

- Systronix
 - Many different HW boards / solutions
 - JStick, JStamp
 - They have a comparison matrix <http://jstik.systronix.com/compare.htm>
- Java Optimized Processor
 - FPGA solution
 - Well... we have already had the best possible introduction to that :-)

The Future

- Sun SPOT (Wireless Sensor Platform)
 - Small Programmable Object Technology
- VM's for small wireless controller devices and wireless sensor networks
- ARM7 CPU, flash memory and SRAM, as well as an 802.15.4 Zigbee wireless radio chip
- The VM is "Squawk"
 - Optimized J2ME VM written almost entirely in Java
 - Run on bare metal => no need for an RTOS
 - Pre-verifies as usual; all symbolic references are resolve etc.
 - Can run multiple suites
 - Can run multiple suites and each suite's class files is optimized for that
 - Prototype VM is 350 kbytes of RAM and Flash combined
- But, look here for the complete text with critiques:
- <http://www.embedded.com/showArticle.jhtml;jsessionid=LQLP2GZKQH0LWQ5NDLOSXHSCJUNN2JVN?articleID=2788101293>



Conclusion

- Before starting out with an embedded Java project there are a great deal of information that need to consumed
- It is not enough to understand the Java language
- The JVM/KVM should be understood together with the boot process etc., just as the C runtime system is understood.
- Lot's of profiles and configurations should be studied
- Lot's of combinations of hardware and software solutions should be studied
 - But all this is business as usual when looking into pros and cons of new technology
- If the Embedded Java development environment where described better by vendors, as seen from a C/C++ programmers perspective, it's likely that more people would take the step
- We get higher productivity, better tools, better vendor independence and so on.

23

www.hyphen-innovation.com

If you have questions or comments to this presentation, please send an email to martin.astradsson@hyphen-innovation.com