The early history

- James Gosling, Sun Microsystems
- Not the usual start for a prog. language
- Consumer electronics, 1991
- Mosaic and the Internet, 1994
- The revolution: applets, 1995
- Since then, many improvements and additions have been made to the language

Why is Java so appealing?

- Platform independent
- Safe
- Easy to learn
- Powerful, well-documented, and easy-to-use libraries to perform many complicated tasks
- During this presentation, we’ll look into each of these qualities, and more
  - Comparison to C++
  - Hands-on activities
Platform independence

• Sun’s motto for Java:
  *write once, run anywhere*

• It’s a great idea, but…
  • how’s it done?
  • what are the drawbacks?

• Wait… so does it mean that Java is an *interpreted* language? Yes, source is compiled into *bytecode*.
• Aren’t interpreted languages inherently slower than compiled ones? Yes.
• Why you should not care so much, though:
  • Java trades speed for
    • platform independence
    • safety (more on this later)
  • Java compilers are pretty darn good anyway
• Still, if you’re *really* worried about speed, you may always use the so-called *just-in-time (JIT)* compilers.
Safe and easy to learn

- The first thing to note here is that these are relative terms.
- In this talk, we’ll compare Java and C++.
- The general consensus is that Java is easier to learn and use than C++, but I’ll let you be the judge of that.

Is Java safer than C++?

Safer than C++?

- What do we mean by “safe” anyway?
- Less prone to programmer mistakes.
- Java achieves better safety than C++ by:
  - providing sandboxes (we won’t talk much about them here)
  - checking every array access for out-of-bounds errors
  - eliminating direct access to pointer operations
  - automatically reclaiming any (heap) memory space not in use (automatic garbage collection)
  - having a less fragile approach to multiple inheritance
  - making every function virtual
  - providing, generally speaking, a simpler syntax than C++

No pointers?

- Some people claim that Java has no pointers… Not true!
- All objects are accessed through references, which are automatically de-referenced.
- However, the pointer nature of these references is hidden from the programmer. Why?
  - Reduced number of pointer-related errors
**Automatic garbage collection**

- Objects are **always** allocated in the heap, using new, as in `Foo f = new Foo();`
- Functions are always allocated in the stack
- The stack is referenced by its allocated in the heap
- Java keeps track of how many valid references exist for each object — when an object has no more references to it, the memory space it occupies in the heap gets reclaimed
- No; it doesn’t mean that you may be sloppy
- Automatic garbage collection has pros and cons
  - Pro: prevents many common memory allocation bugs
  - Con: has a negative impact on your program’s efficiency

---

**No multiple inheritance?**

- C++ inheritance forces the inheritance of both data and behavior (code)
  - That’s a very fragile approach — in order to inherit some behavior your class may have to gain some data as well, even if it’s not really needed
- Java solves that problem and at the same time eliminates the need for multiple inheritances by defining something called an **interface**
  - Interfaces only define the expected behavior of a set of functions, like a contract – no data and no implementation
  - A class may implement as many interfaces as needed
- Of course, regular inheritance between classes is still allowed, but a class may inherit from only one other class — no multiple class inheritance in Java

---

**Functions are always virtual**

- All (non-static) functions in Java follow a late-binding process
  - Which function code is actually executed depends on the actual run-time type of the object on which the function is being called, not on the object’s declared type at compile time
  - In C++, unless one declares a function to be virtual, the code to be executed is decided at compile time
  - Thus, in Java, all (non-static) functions are virtual
  - Late-binding is a little slower but prevents common hard-to-find bugs
Other differences between Java & C++

- (Almost) everything is an object
  - Only primitive types (boolean, char, int, long, float, double) are not objects
- Function arguments are always passed by value
  - Objects are not copied – only their references are
- Neat solution to name collisions (packages)
- No separation between header and implementation
- No operator overloading
- No structs
- No generics (templates) and no enums (constant enumerations) until Java 2, 1.5

A few other nice things about Java

- Inherently multi-threaded
  - Threads are supported at the language level and are also objects
- Much nicer compiler and run-time error messages than C++
- Exception handling is idiomatic – every Sun-written library uses it and does so consistently
- Powerful and easy-to-use libraries for data structures, multi-threading, networking, I/O, graphics, GUI

Other cool stuff

- Javadoc
  - Auto-documenting your code
    - Your comments are nicely formatted into a set of HTML pages
    - C++ has something similar: Doxygen
  - Swing
    - Dynamically pluggable look-and-feel (plaf)
    - Powerful, easy-to-use GUI toolkit
Examples, please!

Hot from the Sun...
- Current release version: Java 2, 1.4.2
- Java 2, version 1.5 beta is out
  - Support for generics (templates)
  - Autoboxing
  - Enhanced for loop
  - Enumerated types
  - Static import
  - C-style formatted input/output
  - Variable arguments
  - and more...
- http://java.sun.com/developer/technicalArticles/releases/j2se15/

References – online
- The creators of Java
  - http://www.java.sun.com
- The Java Developer Connection is a must. In particular, their Tech Tips free mailing list is awesome.
  - https://developer.java.sun.com/
- A cool site to keep in touch with Java news
- This site's free subscription mailing lists are excellent
  - http://www.javaworld.com
References – books I (basics)

- The Java Tutorial: A Short Course on the Basics
- Thinking in Java
- The JFC Swing Tutorial
- Java in a Nutshell
- Java Cookbook
- The Elements of Java Style

References – books II (intermediate)

- Practical Java Programming Language Guide
- Effective Java Programming Language Guide
- Java Pitfalls: Time-Saving Solutions and Workarounds to Improve Programs
- Design Patterns Java Workbook
- GoF’s Design Patterns (C++)

Questions?

I’d like to thank
- all of you for coming
  (and staying !)
- Titus Winters and Dan Berger, for organizing these talks and for inviting me to give this one