CSC207H: Software Design
Lecture 5

Wael Aboelsaadat

wael@cs.toronto.edu

http://ccnet.utoronto.ca/20075/csc207h1y/

Office: BA 4261

Office hours: R 5-7

Acknowledgement: These slides are based on material by Prof. Karen Reid

Tools in a Software House

✓ Programming Languages

✓ Scripting Languages

- Integrated Development Environment (IDE) App
- Profiling Tools

✓ Version Control App (e.g. cvs)

✓ Quality Assurance Framework (e.g. junit)

→ Software Build Management Framework

- Requirements/Feature Tracking App
- Variance Tracking App
- Architecture Tools

Make

How do you rebuild a program?

- javac A.java
- But what if you have many source files?
  - javac *.java doesn't work with sub-directories
  - And will be very (very) slow for large programs
- And what if some rely on others?
  - Suppose Space.java uses Point.java
  - Change Point.java
  - Forget to compile it
  - Compile and run Space.java
  - oops

Automate(!)

- Anything worth repeating is worth automating
- Computers are good at repetitive tasks, so make the computer do it
- Most widely used tool for this is called Make
  - Invented in 1975 by Stuart Feldman when he was a summer student at Bell Labs
- Make's role:
  - Figure out what has changed
  - Work out what is affected by those changes
  - Execute commands to bring things up to date (*e.g.* by recompiling)

Hello make

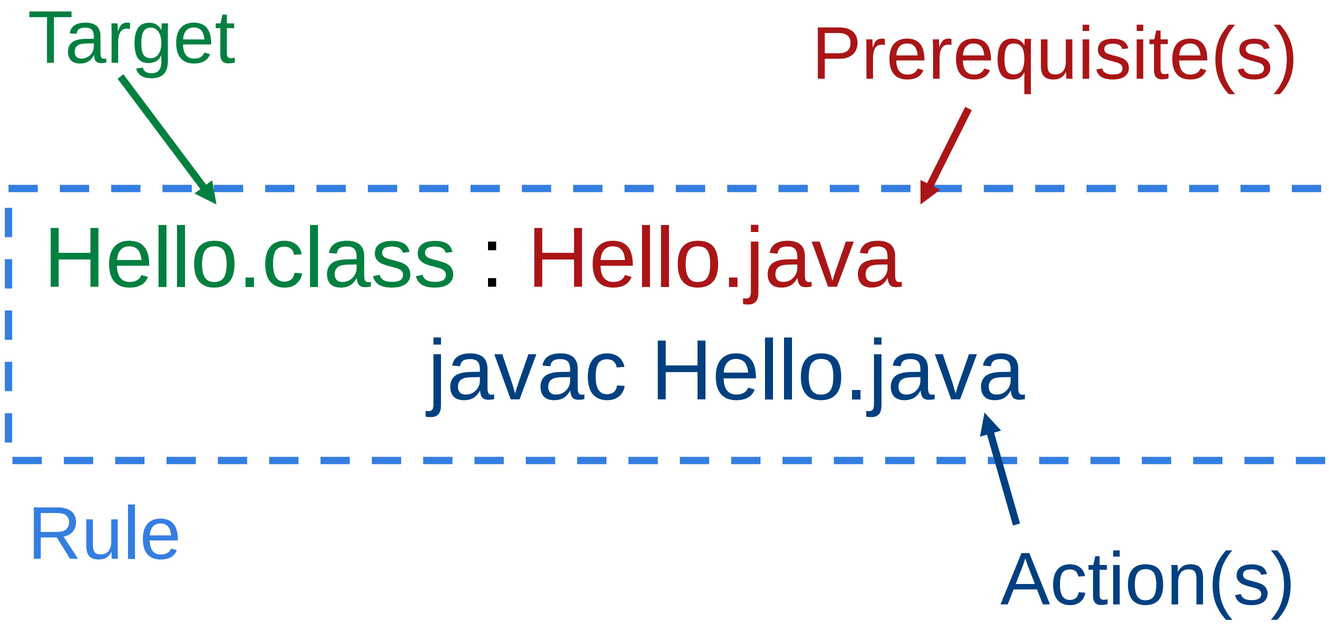
- Put program in Hello.java
- Put the following into a file called **hello.mk**:
- Hello.class : Hello.java
- javac Hello.java

  - Note: that indentation must be a tab

Running make

- Run make -f hello.mk
  - Make sees that Hello.class depends on Hello.java
  - But Hello.class doesn't exist, so Hello.java is compiled
- Run make -f hello.mk again
  - Nothing happens
  - Hello.class is already up to date

Terminology



Target

Prerequisite(s)

```
Hello.class : Hello.java
              javac Hello.java
```

Rule

Action(s)

- May be many prerequisites
- Rule may have many actions (one per line)

How it works

- Make looks at when the target and its prerequisites were last modified
  - It assumes targets are files and checks the dates on the files
- Make does nothing …
  - If the target exists, and
  - Is more recent than all its prerequisites
- Make executes the actions …
  - If the target doesn't exist, or
  - If any prerequisite is more recent than the target

Multiple targets

- # double.mk
- Left.class : Left.java
-         javac Left.java
- Right.class : Right.java
-         javac Right.java
- Run make -f double.mk
  - Only Left.java is compiled
  - Because the first target in the file is the default
- Run this to build Right.class:
  - make -f double.mk Right.class

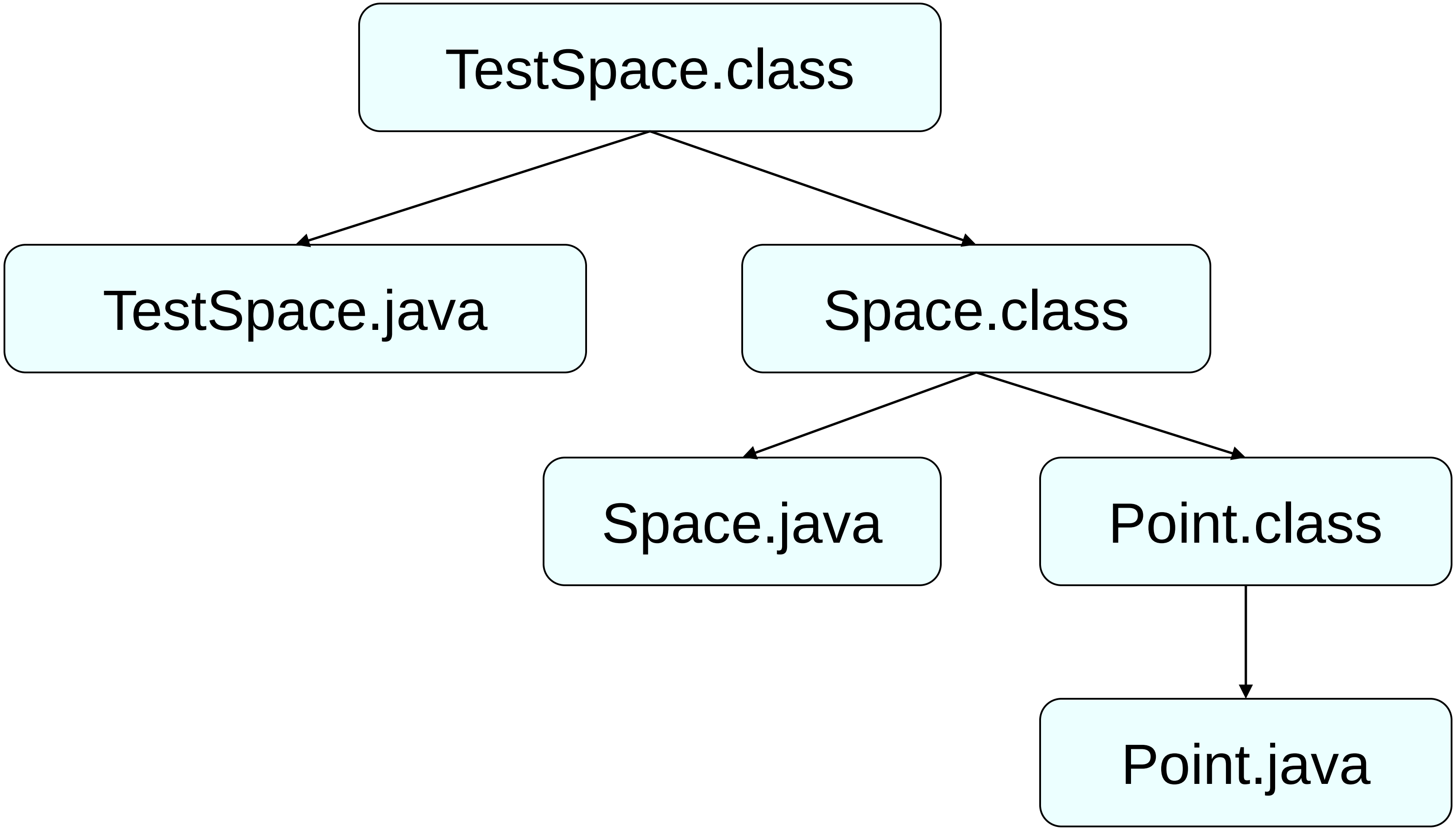
Phony targets

- # all.mk
- all : Left.class Right.class
- Left.class : Left.java
- javac Left.java
- Right.class : Right.java
- javac Right.java
- all is a "phony target"
  - No file called all
  - Never up to date
- make -f all.mk compiles both Java files

Multiple dependencies

- Having targets depend on other targets forces make to do things in a certain order
  - TestSpace.class : TestSpace.java Space.class
  - javac TestSpace.java
  - Space.class : Space.java Point.class
  - javac Space.java
  - Point.class : Point.java
  - javac Point.java

Visualizing dependencies



TestSpace.class

TestSpace.java

Space.class

Space.java

Point.class

Point.java

Avoiding redundancy

- Often want to set options when compiling
  - -classpath to include libraries
  - -d to specify output directory
  - -source 1.5 to specify Java language version
- Anything repeated in two or more places will eventually be wrong in at least one
  - Define variables (usually called "macros" in Make)
  - Warning: syntax is a bit tricky

Macro Example

JC = javac -classpath ".:/usr/jar/junit.jar" -source 1.4

- TestSpace.class : TestSpace.java Space.class
-         ${JC} TestSpace.java

- Space.class : Space.java Point.class
-         ${JC} Space.java

- Point.class : Point.java
-         ${JC} Point.java

Automatic variables

- Make defines variables to represent parts of rules

$@	The target
$<	The first prerequisite
$?	All out-of-date prerequisites
$^	All prerequisites

Automatic variable example

- JC = javac -source 1.4
- TestSpace.class : TestSpace.java Space.class
- @echo "Building" $@
- ${JC} $<
- Space.class : Space.java Point.class
- @echo "Building" $@
- ${JC} $<
- Point.class : Point.java
- @echo "Building" $@
- ${JC} $<

Huh?

- `@echo "Building" $@`

- What is `echo`?
  - A program to print to stdout
- What is `@`?
  - Don’t print the action, just do it

Pattern rules: smarter way to write a make file

- Most files are compiled the same way
  - So write a pattern rule for the general case
  - `%.class : %.java`
  - `                ${JC} $<`
  - Use % to mark the stem of the file's name
  - Like using * in commands in DOS or Unix
- Accumulate extra prerequisites by giving rules without actions
  - *E.g.* Space.class : Point.class

Analysis

- Pro
  - Simple things are simple to do...
  - ...and to read
- Con
  - The syntax is unpleasant
  - Complex things are difficult to read ...
  - ... and even more difficult to debug
  - Not really very portable
- Uses native shell to execute commands
  - Do you use `del` or `rm` to delete files?

Example

```
•   JR = java -enableassertions

•   all : run

•   run : Words.class in1.txt
•           ${JR} Words in1.txt

•   test : Words.class in1.txt out1.txt
•           ${JR} Words in1.txt | diff - out1.txt

•   clean :
•           @rm -f *~ *.class

•   %.class : %.java
•           javac -source 1.4 $<
•   COMPILE_JAVA = javac -classpath ${CSC207PATH} -source 1.4
•   RUN_JAVA     = java  -classpath ${CSC207PATH} -enableassertions

•   test: MorseTests.class
•           ${RUN_JAVA} MorseTests

•   MorseTests.class: \
•           MorseCoder.class DuplicateCodeException.class \
•           UnassignedSymbolException.class InvalidCodeException.class

•   MorseCoder.class: \
•           DuplicateCodeException.class \
•           UnassignedSymbolException.class \
•           InvalidCodeException.class

•   clean :
•           @rm -f *~ *.class *.pyc

•   %.class : %.java
•           @${COMPILE_JAVA} $<
```

Tools in a Software House

- ✓ Programming Languages
- ✓ Scripting Languages
- Integrated Development Environment (IDE) App
- Profiling Tools
- ✓ Version Control App (e.g. cvs)
- ✓ Quality Assurance Framework (e.g. junit)
- ✓ Software Build Management Framework (e.g. make)
- Requirements/Feature Tracking App
- Variance Tracking App
- Architecture Tools

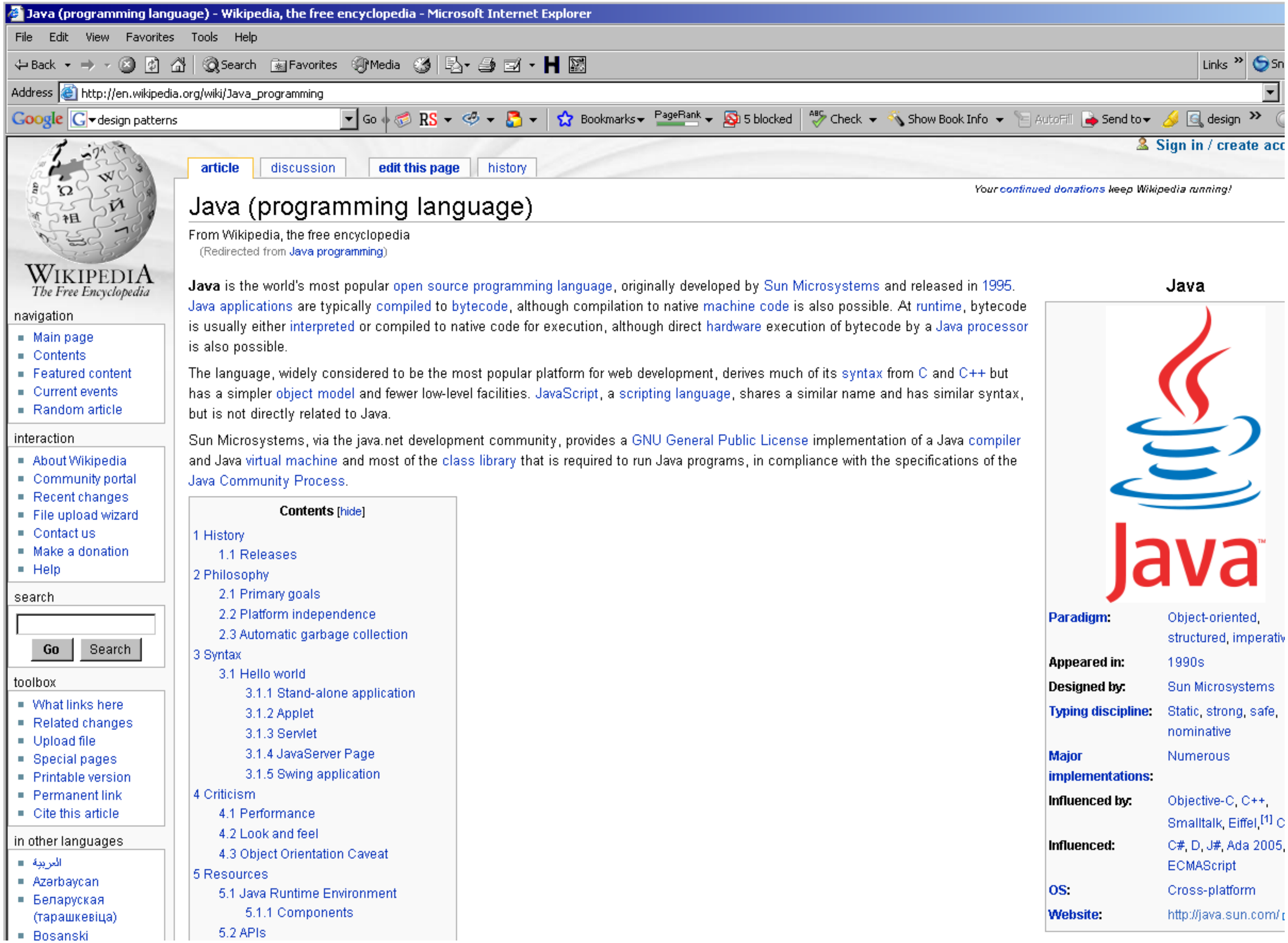
Regular Expressions

Regular Expressions

- A mini-tool supported by all serious Programming/scripting languages

- Can’t live without!

Motivation



Your continued donations keep Wikipedia running!

Java (programming language)

From Wikipedia, the free encyclopedia

(Redirected from Java programming)

Java is the world's most popular open source programming language, originally developed by Sun Microsystems and released in 1995. Java applications are typically compiled to bytecode, although compilation to native machine code is also possible. At runtime, bytecode is usually either interpreted or compiled to native code for execution, although direct hardware execution of bytecode by a Java processor is also possible.

The language, widely considered to be the most popular platform for web development, derives much of its syntax from C and C++ but has a simpler object model and fewer low-level facilities. JavaScript, a scripting language, shares a similar name and has similar syntax, but is not directly related to Java.

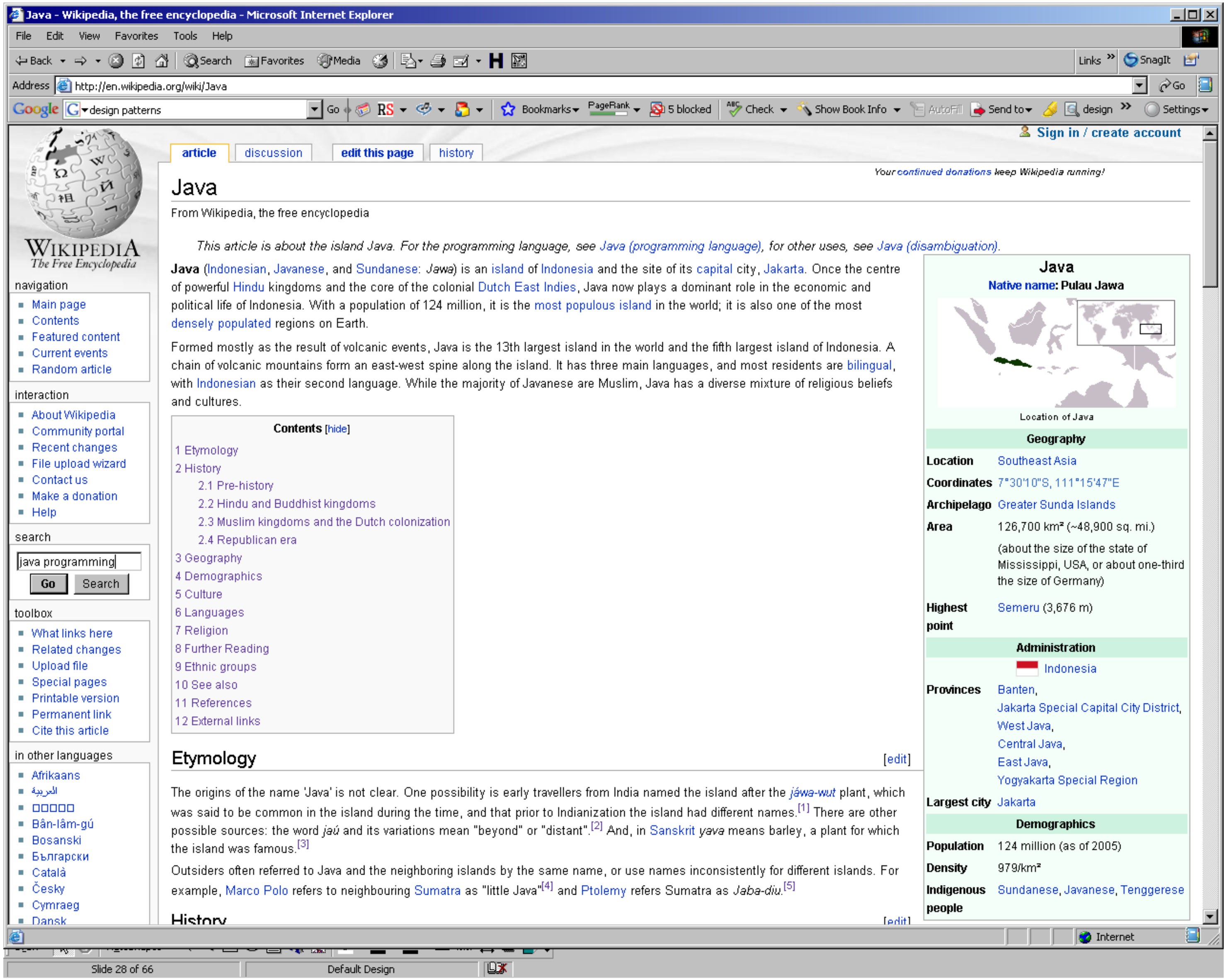
Sun Microsystems, via the java.net development community, provides a GNU General Public License implementation of a Java compiler and Java virtual machine and most of the class library that is required to run Java programs, in compliance with the specifications of the Java Community Process.

Contents [hide]

Java

Paradigm:	Object-oriented, structured, imperativ
Appeared in:	1990s
Designed by:	Sun Microsystems
Typing discipline:	Static, strong, safe, nominative
Major implementations:	Numerous
Influenced by:	Objective-C, C++, Smalltalk, Eiffel,[1] C
Influenced:	C#, D, J#, Ada 2005, ECMAScript
OS:	Cross-platform
Website:	http://java.sun.com/

Motivation



Java - Wikipedia, the free encyclopedia - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Links SnagIt
Address http://en.wikipedia.org/wiki/Java
Go
Google design patterns
Go RS Bookmarks PageRank 5 blocked Check Show Book Info AutoFill Send to design Settings
Sign in / create account
article discussion edit this page history
Internet
Slide 28 of 66 Default Design

Your continued donations keep Wikipedia running!

Wikipedia
The Free Encyclopedia

navigation
- Main page
- Contents
- Featured content
- Current events
- Random article

interaction
- About Wikipedia
- Community portal
- Recent changes
- File upload wizard
- Contact us
- Make a donation
- Help

search

java programming

Go Search

toolbox
- What links here
- Related changes
- Upload file
- Special pages
- Printable version
- Permanent link
- Cite this article

in other languages
- Afrikaans
- العربية
- □□□□□
- Bân-lâm-gú
- Bosanski
- Български
- Català
- Česky
- Cymraeg
- Dansk

Java

From Wikipedia, the free encyclopedia

This article is about the island Java. For the programming language, see Java (programming language), for other uses, see Java (disambiguation).

Java (Indonesian, Javanese, and Sundanese: *Jawa*) is an island of Indonesia and the site of its capital city, Jakarta. Once the centre of powerful Hindu kingdoms and the core of the colonial Dutch East Indies, Java now plays a dominant role in the economic and political life of Indonesia. With a population of 124 million, it is the most populous island in the world; it is also one of the most densely populated regions on Earth.

Formed mostly as the result of volcanic events, Java is the 13th largest island in the world and the fifth largest island of Indonesia. A chain of volcanic mountains form an east-west spine along the island. It has three main languages, and most residents are bilingual, with Indonesian as their second language. While the majority of Javanese are Muslim, Java has a diverse mixture of religious beliefs and cultures.

Contents [hide]

Java

Native name: Pulau Jawa

Location of Java

Geography	
Location	Southeast Asia
Coordinates	7°30′10″S, 111°15′47″E
Archipelago	Greater Sunda Islands
Area	126,700 km² (~48,900 sq. mi.) (about the size of the state of Mississippi, USA, or about one-third the size of Germany)
Highest point	Semeru (3,676 m)
Administration	
	Indonesia
Provinces	Banten, Jakarta Special Capital City District, West Java, Central Java, East Java, Yogyakarta Special Region
Largest city	Jakarta
Demographics	
Population	124 million (as of 2005)
Density	979/km²
Indigenous people	Sundanese, Javanese, Tenggerese

Etymology [edit]

The origins of the name 'Java' is not clear. One possibility is early travellers from India named the island after the *jáwa-wut* plant, which was said to be common in the island during the time, and that prior to Indianization the island had different names.[1] There are other possible sources: the word *jaú* and its variations mean "beyond" or "distant".[2] And, in Sanskrit *yava* means barley, a plant for which the island was famous.[3]

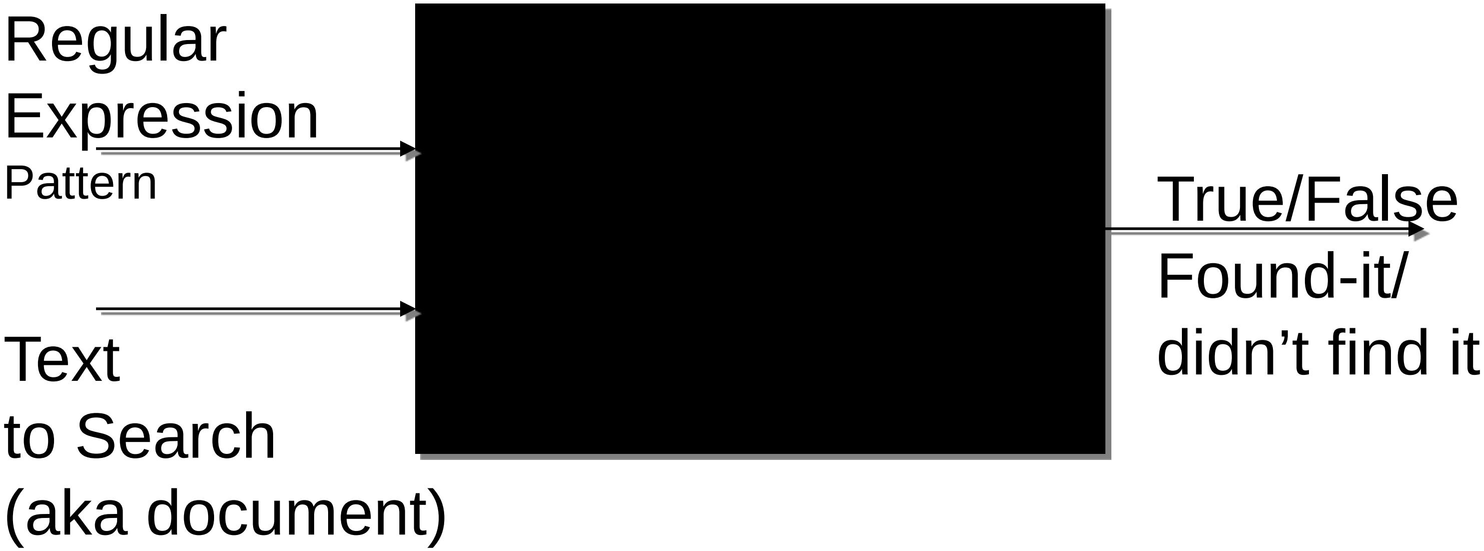
Outsiders often referred to Java and the neighboring islands by the same name, or use names inconsistently for different islands. For example, Marco Polo refers to neighbouring Sumatra as "little Java"[4] and Ptolemy refers Sumatra as *Jaba-diu*.[5]

History [edit]

Motivation: it’s all about searching in text

- Java
- Java and language
- “Java language”
- Java and language and programming
- Java and language or programming
- Java but not Indonesia

Regular Expression Matcher



Regular Expression Pattern

Text to Search (aka document)

True/False Found-it/ didn’t find it

Simple RE Patterns

Pattern	**Explanation**	**Matches**	**Doesn’t Match**
a	either a or not a!	a	b, c, d, X
ab	either ab or not ab!	ab	abc,a,b
a*	* is for zero or more	empty-string, a,aa	b, bb
b+	+ is for one or more	b, bb	ac, aa
b?c	One or zero	c, bc	abc
[abc]	one from a set	a,b, c	ab, bc
[a-c]	Abbreviation	a, b, c	ab, bc
[abc]*	Combination	empty-string, acbccb, bb,ca	abcd
[abc]+	Combination	acbccb, bb,ca	empty-string

Anchoring

- Force the position of match
  - ^ matches the beginning of the line
  - $ matches the end
  - Neither consumes any characters.

pattern	**text**	**result**
b+	abbc	Matches
^b+	abbc	Fails (no b at start)
^b+	bbc	Matches
b+$	cb	Matches
^a*$	aabaa	Fails (not all a's)

Escaping

- Match actual ^ and $ using escape sequences \^ and \$
- Match actual + and * using escape sequences \+ *
- Be careful with back slashes
- Use escapes for other characters:
  - \t is a tab character
  - \n is a newline

Character sets

- Use escape sequences for common character sets

\d	Digits	[0-9]
\w	Word	[a-zA-Z0-9_]
\s	Space	[\t\n\r]
.	Anything except end of line	[^\n]

- Note the notation [^abc] means “anything not in the set”

RE Patterns: more high-level..

Patterns:	**Matches**	**Doesn’t Match**
a	a	b
ab	ab	aa
a | b	'a', 'b'	ab
ab|cd	'ab', 'cd',	ad, aab
a(bc|de)f	'abcf', 'adef'	af

Compiling

- Regular expression library compiles patterns into more concise form for matching
- Can improve performance by doing this once, and re-using the compiled pattern

Regular expressions in Java

- The java.util.regex package contains:
  - Pattern: a compiled regular expression
  - Matcher: the result of a match

```
• public String matchMiddle(String data) {
•     String result = null;
•     Pattern p = Pattern.compile("a(b|c)d");
•     Matcher m = p.matcher(data);
•     if (m.matches()) {
•         result = m.group(1);
•         }
•     return result;
• }
```

How to use in Python

- Import the re module
- Use re.search(pattern, text)

```
•  import sys, re
•  pat = sys.argv[1]
•  for text in sys.argv[2:]:
•      if re.search(pat, text):
•          result = "FOUND"
•      else:
•          result = "NOT FOUND"
•      print pat, text, result
```

- **$ testMatch "a[bc]*" b ab accb add**

```
a[bc]* b NOT FOUND
a[bc]* ab FOUND
a[bc]* accb FOUND
a[bc]* add FOUND
```

Match Objects

- Results of re.search() is a match object
  - mo.group() returns string that matched
  - mo.start() and mo.end() are the match's location

- mo = re.search("b+" , "abbcb")
- print mo.group(), mo.start(), mo.end()

- ***bb 1 3***

Python: functions & classes

Class members (new…)

- Variables defined directly in the class belong to the class
  - Not related to any self instance
  - Like static in Java
- Nothing equivalent for methods
  - Concept is easy
  - Coming up with a simple syntax has proven difficult
  - We'll see later that it is possible to have methods that are independent of classes: *functions*

Example

- A class variable:

```
• class Tracker:
•     numCreated = 0
•     def __init__(self):
•         Tracker.numCreated += 1

• t1 = Tracker()
• t2 = Tracker()
• print Tracker.numCreated
```

- ***Output: 2***

Creating and loading modules

- Any Python file can be loaded as a module using import module
  - File called xyz.py becomes module xyz
- Statements are executed as module loads
  - Libraries typically just define constants and functions
- Module contents referred to as module.content
  - E.g. sys.argv
- Can also use
  - from module import name1, name2
  - from module import *

Module: example

- # stuff.py
- value = 123
- def printVersion():
-     print "Stuff Version 2.2 "

- # loader.py
- import stuff
- print stuff.value
- stuff.printVersion()

- *$ python stuff.py*
- *$ python loader.py*
- ***123***
- ***Stuff Version 2.2***

Modules: loading versus running

- Special variable __name__ is module's name
  - Set to "__main__" when run from the command line
  - Set to the module's name when loaded by something else
- Often used to include self-tests in module
  - Tests use assert when module run directly

Module: self-test

```
•  class C:
•  def double(self,val):
•      return val * 2

•  if __name__ == '__main__':
•      print "testing C.double"
•      c = C()
•      assert c.double(0) == 0
•      assert c.double('a') == 'aa'
•      assert c.double([1]) == [1, 1]
•      print "tests passed"
```

Python Sequences

Strings

- An immutable sequence of characters
- No separate character type
- Immutable: cannot be modified in place
  - Safety
  - Efficiency

String indexing

- element = "boron"
- i = 0
- while i < len(element):
- print element[i]
- i += 1
- ***b***
- ***o***
- ***r***
- ***o***
- ***n***

Negative string indices

- Negative indices count backward from the end of the string
  - x[-1] is the last character
  - x[-2] is the second-last character

- Example:
  - val = "carbon"
  - print val[-2], val[-4], val[-6]

  - ***o r c***

Slicing

- a[start:end] is the elements of a from start up to (but not including) end
  - Think of the loop for (i = 0; i < n; i++)

- val = "helium"
- print val[1:3], val[:2], val[4:]
- print val[-1:1]

- **el he um**
- **# the empty string**

Bounds

- Out-of-range slice indices treated as though they ended at the end of the range
- Single item access: bounds *always* checked; out-of-bounds index results in an error:
  - val = "helium"
  - print val[1:22]
  - x = val[22]

  - **elium**
  - **IndexError: string index out of range**

Slicing creates a new object

- A slice is a new list
  - Not an alias for subsection of existing list

  - x = ["a", "b", "c", "d"]
  - y = x[0:2]
  - y[0] = 123
  - print y
  - print x

  - **[123, "b"]**
  - **["a", "b", "c", "d"]**

Splicing

- *Splice*: to add a piece (possibly in the middle) to a piece of tape or string
- Assigning to a slice splices the lists
  - Replace the (possibly empty) section of list with a (possibly empty) list

  - x = ["a", "b", "c", "d"]
  - x[1:1] = ["x", "y", "z"]
  - print x

  - **["a", "x", "y", "z", "b", "c", "d"]**

More on splicing

- Inserted object (spliced in) must be a list
  - x = ["a", "b", "c"]
  - x[1:2] = "z"
  - ***TypeError: must assign list (not ‘str’) to slice***
- Splicing in the empty list removes elements
  - x = ["a", "b", "c", "d"]
  - x[1:3] = []
  - print x
  - ***["a", "d"]***

Python Functions

More on functions: memory

- Function arguments always copied
  - Means structures are aliased
  - Just as in Java
- def mutate(x, y):
- x = 0
- y[0] = 0
- a = 1
- b = [1, 1, 1]
- mutate(a, b)
- print a, b # ***1, [0, 1, 1]***

Default argument values

- Can provide defaults for arguments
- Arguments without defaults must come first
  - def withTax(val, percent=14):
  -     return val * (1.0 + percent/100.0)
  - print withTax(10.00)   # default
  - print withTax(10.00, 6) # explicit
  - ***11.4***
  - ***10.6***

Named arguments

- Can pass arguments in any order using names
  - def show(first, second):
  -     print first, second
  - show(1, 2)
  - show(second=9, first=0)
  - ***1 2***
  - ***0 9***