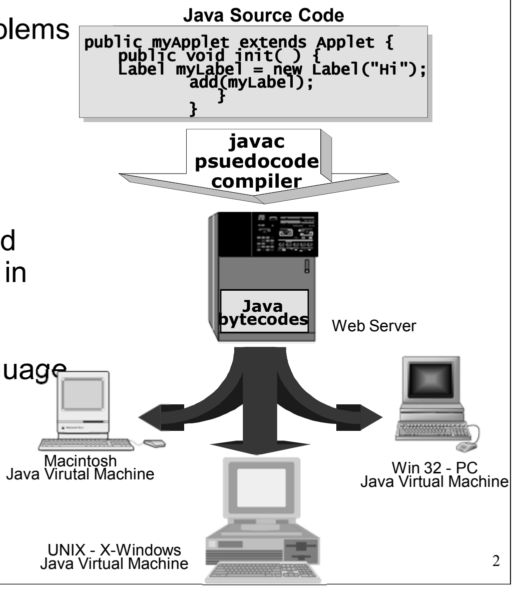
Java Programming Basics: Identifiers, Types, Variables, Operators, and Control Flow

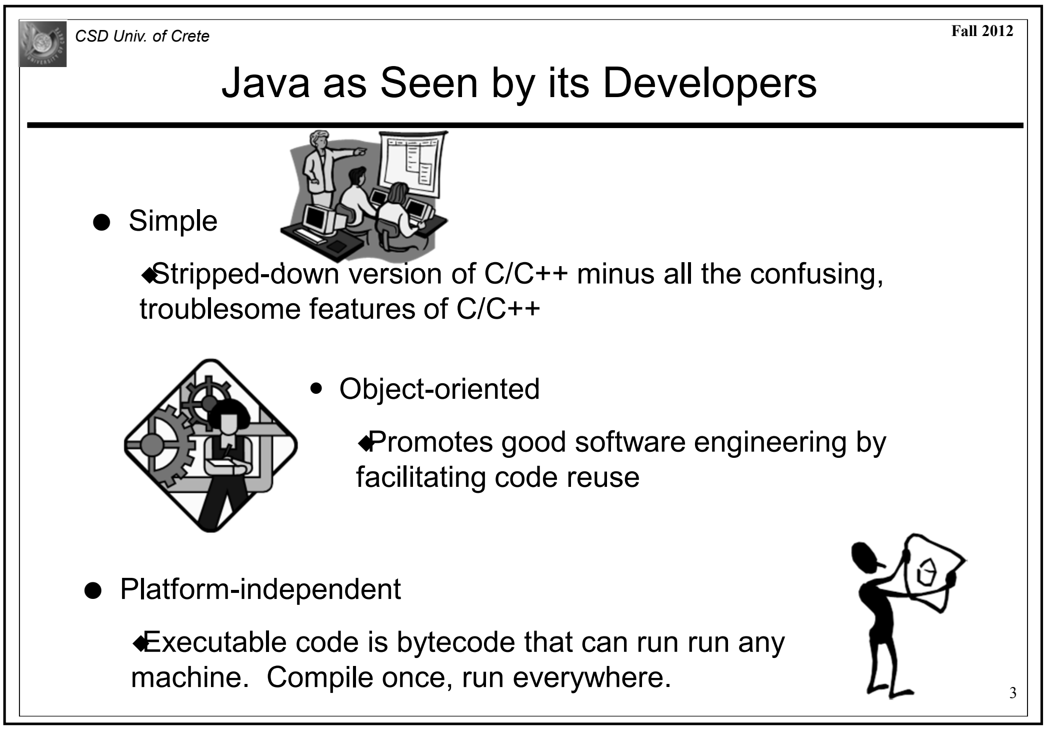
So, Where’s the Java?

- Developers of Java tried to "fix" problems of earlier languages:
  - ◆too many different platforms (including "appliances" such as phones, toasters, etc.)
  - ◆too many incompatibilities
  - ◆too much machine orientation and detail, without objects being built in
  - ◆difficult to use
- Java started as a programming language for embedded systems (toasters, microwave ovens, washers, etc.)
  - ◆needed to be portable
  - ◆had to be reliable
- Web interest came along later



Java Source Code

```
public myApplet extends Applet {
    public void init( ) {
    Label myLabel = new Label("Hi");
          add(myLabel);
              }
          }
```



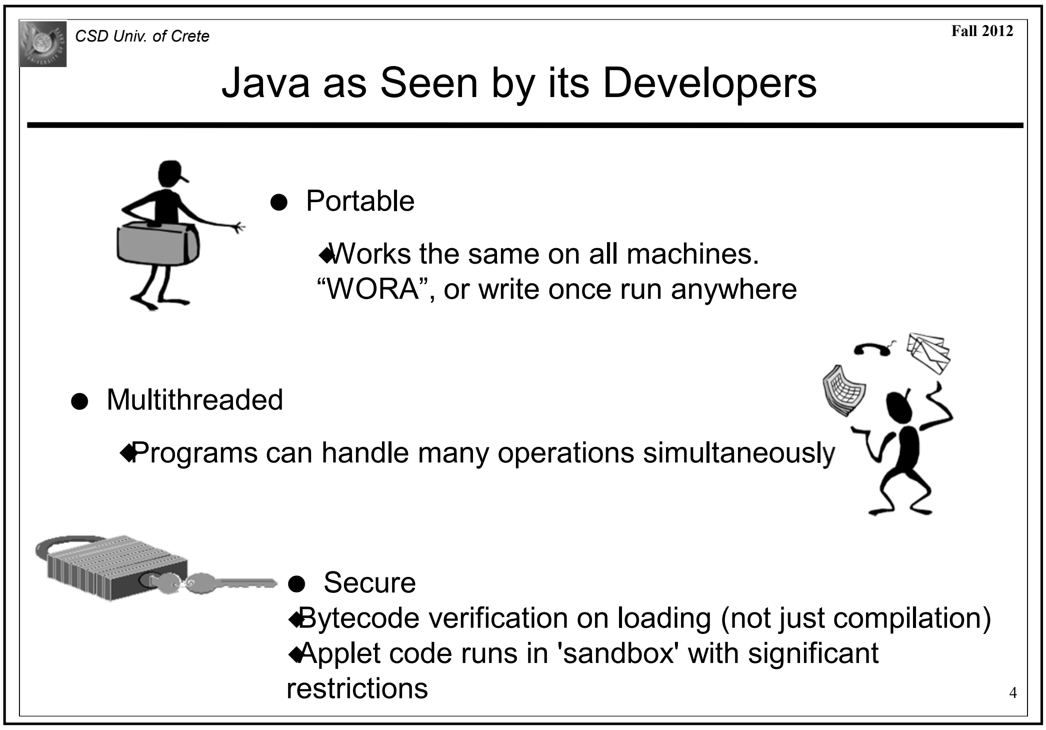
CSD Univ. of Crete

Fall 2012

Java as Seen by its Developers

- Simple
  - Stripped-down version of C/C++ minus all the confusing, troublesome features of C/C++
- Object-oriented
  - Promotes good software engineering by facilitating code reuse
- Platform-independent
  - Executable code is bytecode that can run run any machine. Compile once, run everywhere.

3



CSD Univ. of Crete

Fall 2012

Java as Seen by its Developers

- Portable
  - Works the same on all machines. “WORA”, or write once run anywhere
- Multithreaded
  - Programs can handle many operations simultaneously
- Secure
  - Bytecode verification on loading (not just compilation)
  - Applet code runs in 'sandbox' with significant restrictions

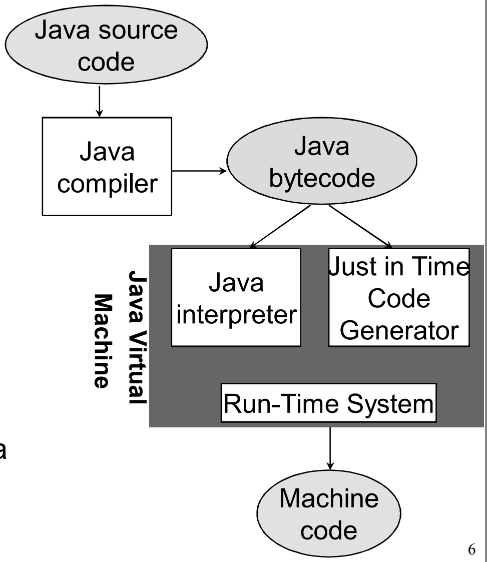
4

How is Java Different from other Languages

- Less than you think:
  - Java is an imperative language (like C++, Ada, C, Pascal)
  - Java is interpreted (like LISP, APL)
  - Java is garbage-collected (like LISP, Eiffel, Modula-3)
  - Java can be compiled (like LISP)
  - Java is object-oriented (like C++, Ada, Eiffel)
- A successful hybrid for a specific-application domain
- A reasonable general-purpose language for non-real-time applications
- Work in progress: language evolving rapidly

How Java works?

- The Java compiler translates Java source code into a special representation called bytecode
  - Java bytecode is not the machine language for any traditional CPU
- Another software tool, called an interpreter translates bytecode into machine language and executes it
  - The JVM is a software layer that provides translation between Java byte codes and the native operating system



Bytecode

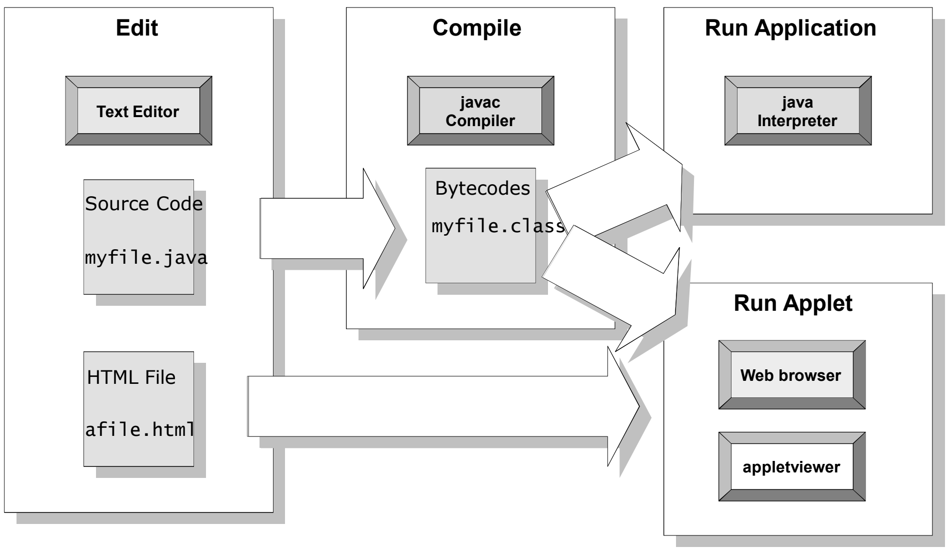
- Like Machine code but machine independent
  - ◆VM interprets each bytecode and executes it in the system's machine opcode
  - ◆VM inner loop

```
do {
   fetch an opcode byte
   execute an action depending
     on the value of the opcode
} while (there is more to do);
```

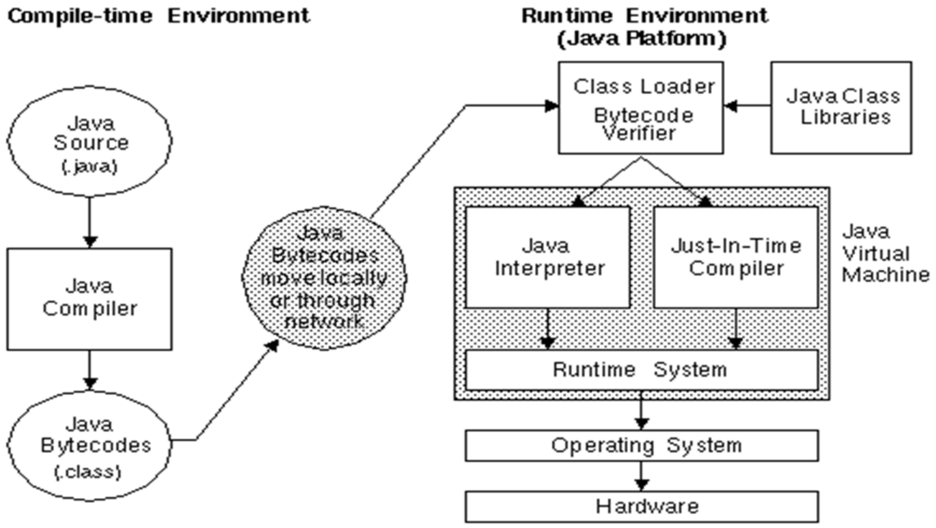
- Each VM is machine dependent
- Just in Time (JIT) code generator can improve the performance of Java Applications by compiling bytecode to machine code before execution

```
AL_CODE: Method 1
    Method="HelloWorld.main"(#24)
    Signature="([Ljava/lang/String;)V"(#26)
    Access=public,static (0x0009)
    Attribute Count=1
    Attribute="Code"(#20)
                 Length=51
                 Max Stack=2
                 Max Locals=1
                Code Length=9
0x00000000 B20008getstatic
                      (#8) java/lang/System.out
0x00000003 1201  ldc
                      (#1) "Hello World"(#34)
0x00000005 B60007invokevirtual
              (#7) java/io/PrintStream.println
0x00000008 B1    return
    Exception Handler Entries=0
    Attribute="LineNumberTable"(#17)
                 Length=6
                 Entry Count=1
               Start=0x00000000 Line Number=5
    Attribute="LocalVariableTable"(#15)
                 Length=12
                 Entry Count=1
    Start=0x00000000 Effective Length=9
                 Slot=0
    Name="args"(#35)
     Signature="[Ljava/lang/String;"(#36)
```

The Java Development Process



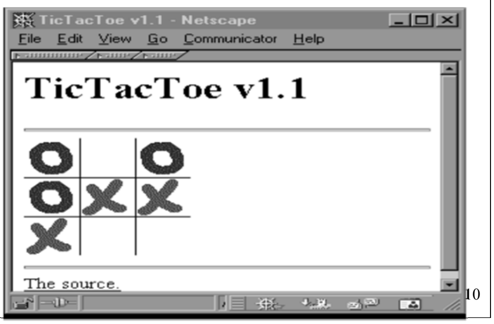
Java Compile vs RunTime Environment



Using Java with HTML: Applets

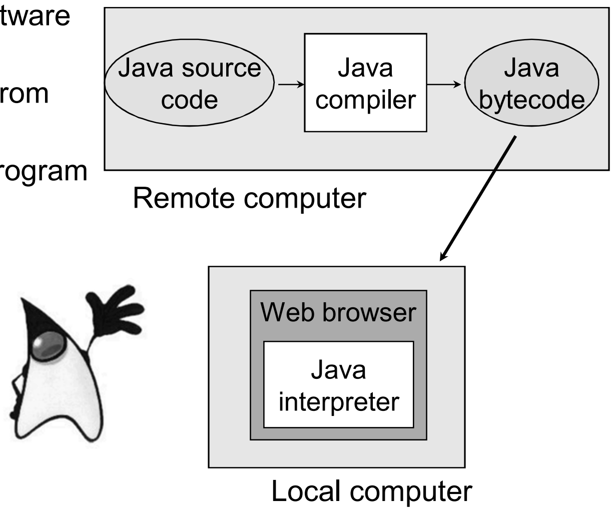
- A Java applet is a Java program that is intended to be sent across a network and executed using a Web browser
- Links to applets can be embedded in HTML documents
  - ◆The <APPLET> Tag

```
<HTML>
<HEAD>
<TITLE>TicTacToe v 1.1</TITLE></HEAD>
<BODY>
<H1>TicTacToe v 1.1 </H1>
<APPLET CODEBASE="/Java/demo/TicTacToe"
 CODE="TicTacToe.class"
 WIDTH="320" HEIGHT="240">
</APPLET>
... </BODY>
</HTML>
```


TicTacToe v1.1 - Netscape
File Edit View Go Communicator Help
TicTacToe v1.1
The source.

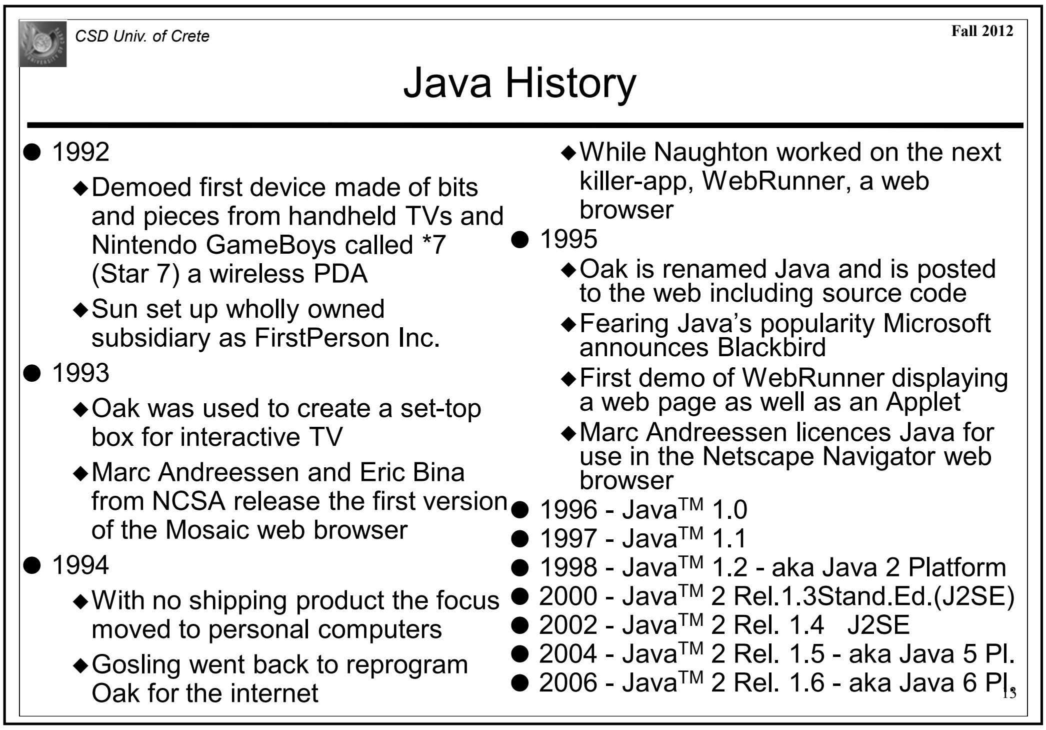
Java Applets

- Most Internet Browser software contains a JVM
  - ◆Load java byte codes from the remote computer
  - ◆Run locally the Java Program in a Browser Window



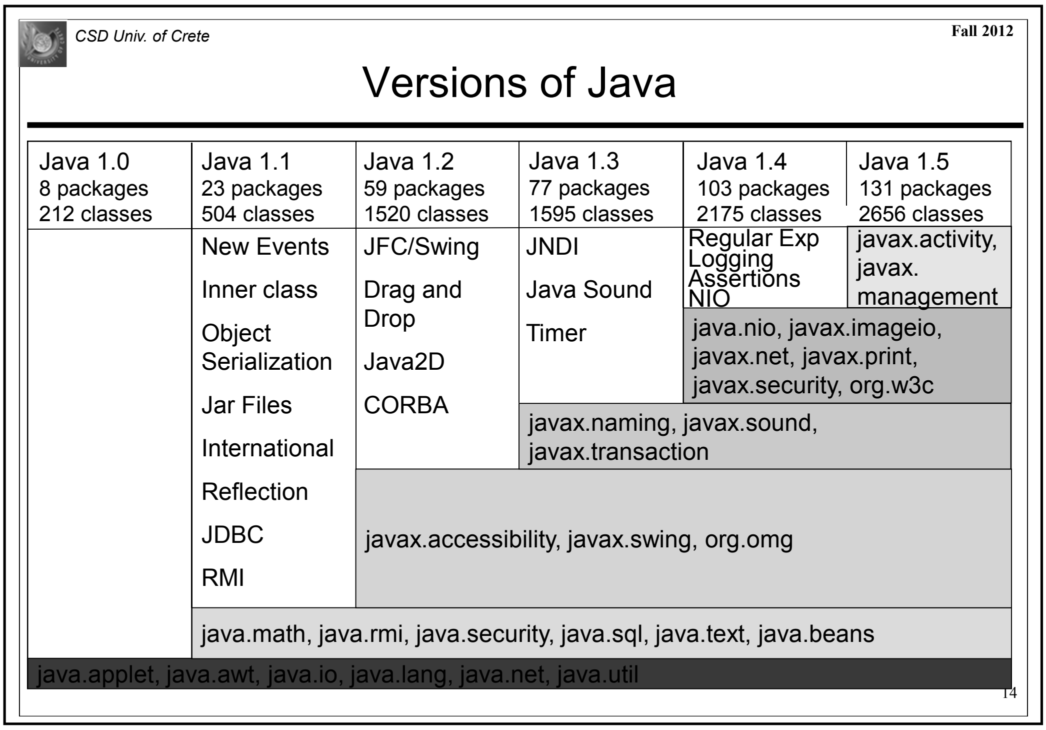
Java History

- 1990
  - ◆Patrick Naughton threatens to leave Sun
  - ◆Scott McNeally CEO of Sun asks Naughton to write up list of problems with Sun and what should be done
    - “Tell me what you would do if you were God.”
  - ◆Naughton’s suggestions
    - hire an artist to pretty up Sun’s uninspired interfaces
    - pick a single programming toolkit
    - focus on a single windows technology
    - lay off just about everybody in the existing windows group
  - ◆McNeally agrees to Naughton’s suggestions and gives Naughton, James Gosling and Mike Sheridan $1 million and one year to deliver
  - ◆The group was code named The Green Project
  - ◆Goal was to develop a system that was compact and simple - driven by consumer electronics
- 1991
  - ◆Green was a solution looking for a problem
  - ◆Gosling realised that C++ wasn’t reliable enough for communicating consumer electronics and developed the language Oak
  - ◆Naughton was interested in the user interface and developed multimedia animations to work with Oak
  - ◆The Green team’s ambition was to build a device that was an interface to cyberspace in colour and 3-D written in Oak



Java History

- 1992
  - ◆Demoed first device made of bits and pieces from handheld TVs and Nintendo GameBoys called *7 (Star 7) a wireless PDA
  - ◆Sun set up wholly owned subsidiary as FirstPerson Inc.
- 1993
  - ◆Oak was used to create a set-top box for interactive TV
  - ◆Marc Andreessen and Eric Bina from NCSA release the first version of the Mosaic web browser
- 1994
  - ◆With no shipping product the focus moved to personal computers
  - ◆Gosling went back to reprogram Oak for the internet
  - ◆While Naughton worked on the next killer-app, WebRunner, a web browser
- 1995
  - ◆Oak is renamed Java and is posted to the web including source code
  - ◆Fearing Java's popularity Microsoft announces Blackbird
  - ◆First demo of WebRunner displaying a web page as well as an Applet
  - ◆Marc Andreessen licences Java for use in the Netscape Navigator web browser
- 1996 - Java^{TM} 1.0
- 1997 - Java^{TM} 1.1
- 1998 - Java^{TM} 1.2 - aka Java 2 Platform
- 2000 - Java^{TM} 2 Rel.1.3Stand.Ed.(J2SE)
- 2002 - Java^{TM} 2 Rel. 1.4 J2SE
- 2004 - Java^{TM} 2 Rel. 1.5 - aka Java 5 Pl.
- 2006 - Java^{TM} 2 Rel. 1.6 - aka Java 6 Pl.

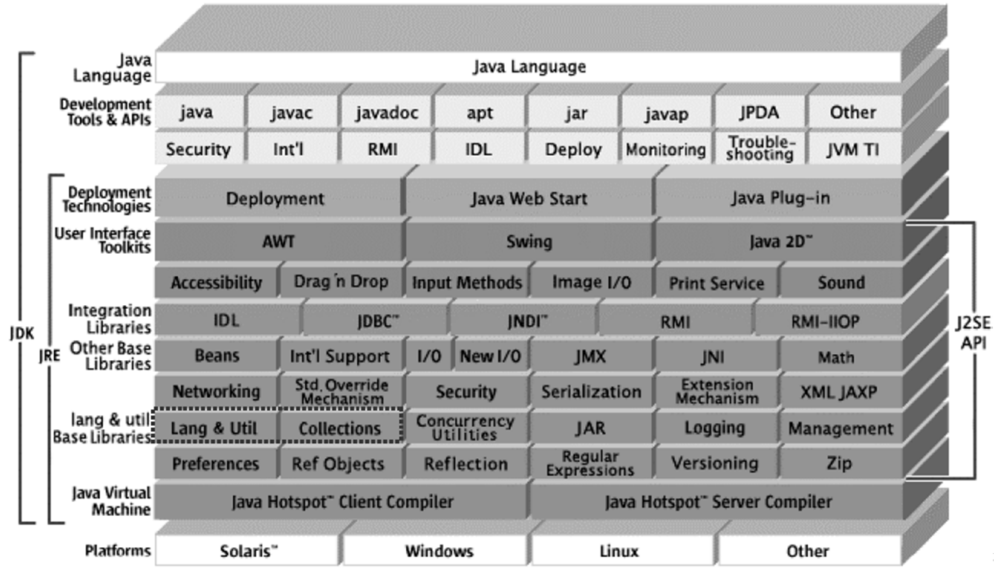


Versions of Java

<table>
<tr><td>Java 1.0
8 packages
212 classes</td><td>Java 1.1
23 packages
504 classes</td><td>Java 1.2
59 packages
1520 classes</td><td>Java 1.3
77 packages
1595 classes</td><td>Java 1.4
103 packages
2175 classes</td><td>Java 1.5
131 packages
2656 classes</td></tr>
<tr><td rowspan="5"></td><td rowspan="4">New Events
Inner class
Object Serialization
Jar Files
International
Reflection
JDBC
RMI</td><td rowspan="3">JFC/Swing
Drag and Drop
Java2D
CORBA</td><td rowspan="2">JNDI
Java Sound
Timer</td><td>Regular Exp
Logging
Assertions
NIO</td><td>javax.activity,
javax. management</td></tr>
<tr><td colspan="2">java.nio, javax.imageio, javax.net, javax.print, javax.security, org.w3c</td></tr>
<tr><td colspan="3">javax.naming, javax.sound, javax.transaction</td></tr>
<tr><td colspan="4">javax.accessibility, javax.swing, org.omg</td></tr>
<tr><td colspan="5">java.math, java.rmi, java.security, java.sql, java.text, java.beans</td></tr>
<tr><td colspan="6">java.applet, java.awt, java.io, java.lang, java.net, java.util</td></tr>
</table>

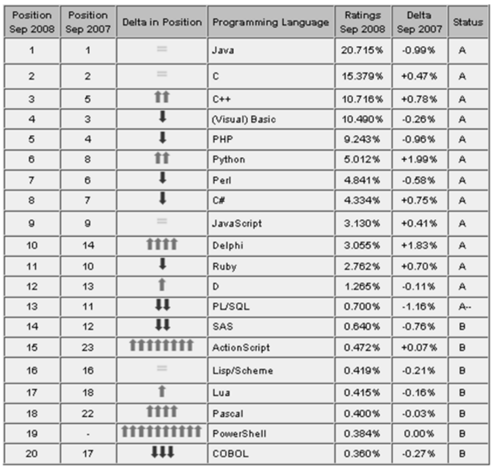
14

J2SE 5.0: Java 2 Platform Standard Edition 5.0



Java Market Facts

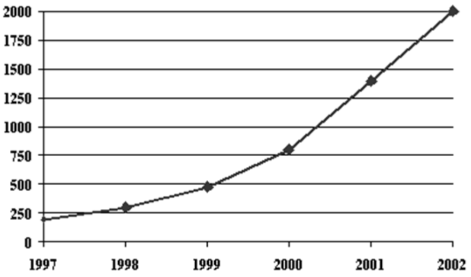
TIOBE Programming Community



Position Sep 2008	Position Sep 2007	Delta in Position	Programming Language	Ratings Sep 2008	Delta Sep 2007	Status
1	1	=	Java	20.715%	-0.99%	A
2	2	=	C	15.379%	+0.47%	A
3	5	↑↑	C++	10.716%	+0.78%	A
4	3	↓	(Visual) Basic	10.490%	-0.26%	A
5	4	↓	PHP	9.243%	-0.96%	A
6	8	↑↑	Python	5.012%	+1.99%	A
7	6	↓	Perl	4.841%	-0.58%	A
8	7	↓	C#	4.334%	+0.75%	A
9	9	=	JavaScript	3.130%	+0.41%	A
10	14	↑↑↑↑	Delphi	3.055%	+1.83%	A
11	10	↓	Ruby	2.762%	+0.70%	A
12	13	↑	D	1.265%	-0.11%	A
13	11	↓↓	PL/SQL	0.700%	-1.16%	A--
14	12	↓↓	SAS	0.640%	-0.76%	B
15	23	↑↑↑↑↑↑↑↑	ActionScript	0.472%	+0.07%	B
16	16	=	Lisp/Scheme	0.419%	-0.21%	B
17	18	↑	Lua	0.415%	-0.16%	B
18	22	↑↑↑↑	Pascal	0.400%	-0.03%	B
19	-	↑↑↑↑↑↑↑↑↑↑↑	PowerShell	0.384%	0.00%	B
20	17	↓↓↓	COBOL	0.360%	-0.27%	B

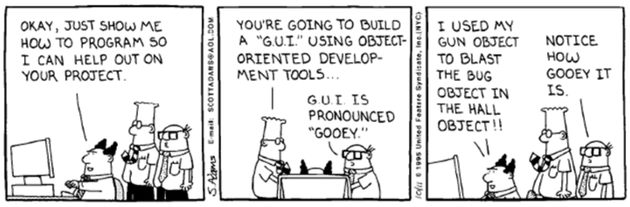
Source: www.tiobe.com September 2008

Java Software Investments in $US



Source: IDC 2000

The Java Phenomena: Myths and Reality

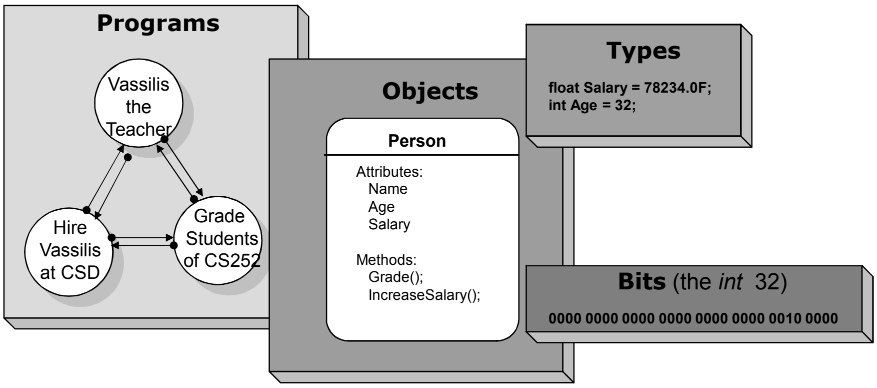

OKAY, JUST SHOW ME HOW TO PROGRAM SO I CAN HELP OUT ON YOUR PROJECT.
YOU'RE GOING TO BUILD A "G.U.I." USING OBJECT-ORIENTED DEVELOPMENT TOOLS...
G.U.I. IS PRONOUNCED "GOOEY."
I USED MY GUN OBJECT TO BLAST THE BUG OBJECT IN THE HALL OBJECT!!
NOTICE HOW GOOEY IT IS.

Dilbert copyright United Media. Used with permission.

- ● Java is Simple
  - ◆Wrong: Java is an advanced programming language (PL)
    - Programming is NOT simple (in any language !!!)
- ● Java Runs on all Platforms
  - ◆Well …Cross-platform code can be achieved, but you must test on all platforms you will deliver on
- ● Java is only for the Internet
  - ◆Wrong again: Java is steadily gaining as a general purpose PL
    - Current systems are about 20% slower than C++
    - Upcoming releases claim to lower or eliminate that gap (10%)
    - No Language, No Operating System, No Platform is right for EVERY Application

Overview of Java Programs

Java's "Building Blocks"



Programs

Vassilis the Teacher

Hire Vassilis at CSD

Grade Students of CS252

Objects

Person

Attributes:
Name
Age
Salary

Methods:
Grade();
IncreaseSalary();

Types

float Salary = 78234.0F;
int Age = 32;

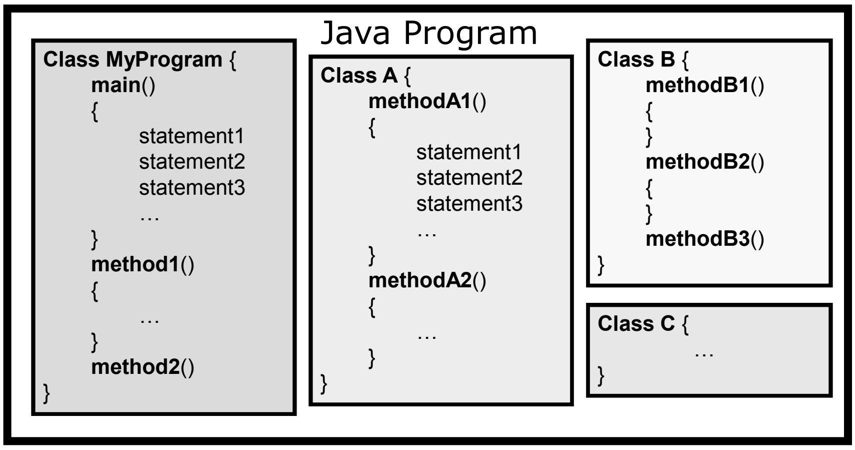
Bits (the *int* 32)

0000 0000 0000 0000 0000 0000 0010 0000

Java's "Building Blocks"

- ● Java programs relies on software components called objects
  - ◆An object contains both data and behavior
  - ◆An object is defined by a class
- ● A program is made up of one or more classes
  - ◆A class contains one or more methods
  - ◆A method contains program statements
- ● Statements are
  - ◆Variable declarations: primitive data types and classes
  - ◆Operations: arithmetic, logical, bit-level, class access
  - ◆Control structures: selection, looping, etc.
  - ◆Object messages: i.e., calls to methods
- ● A Java application always executes the main method

Programs: Classes, Methods and Statements



Java Program

```
Class MyProgram {
      main()
      {
            statement1
            statement2
            statement3
            …
      }
      method1()
      {
            …
      }
      method2()
}
```

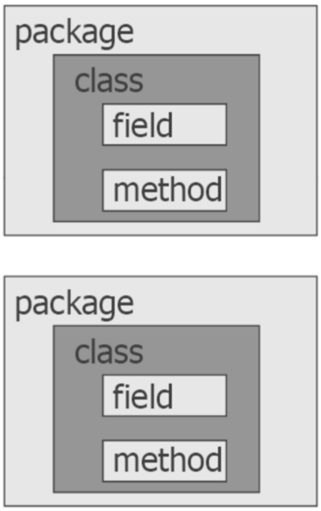
```
Class A {
      methodA1()
      {
            statement1
            statement2
            statement3
            …
      }
      methodA2()
      {
            …
      }
}
```

```
Class B {
      methodB1()
      {
      }
      methodB2()
      {
      }
      methodB3()
}
```

```
Class C {
            …
}
```

Encapsulation and Packages

- Every field, method belongs to a class
- Every class is part of some package
  - The default package is used unless an other package is specified
  - The name of a package is a sequence of names, separated by ”.” e.g., ”java.lang”
  - The fully qualified name of a class is the name of the package followed by the a ”.”followed by the name of the class. The fully qualified name of class String is ”java.lang.String”
  - A package does not declare which classes belong in it. Instead a class define which package it belong to. This is done by the package declaration in a sourcefile, e.g., package gr.uoc.csd.hy252.example



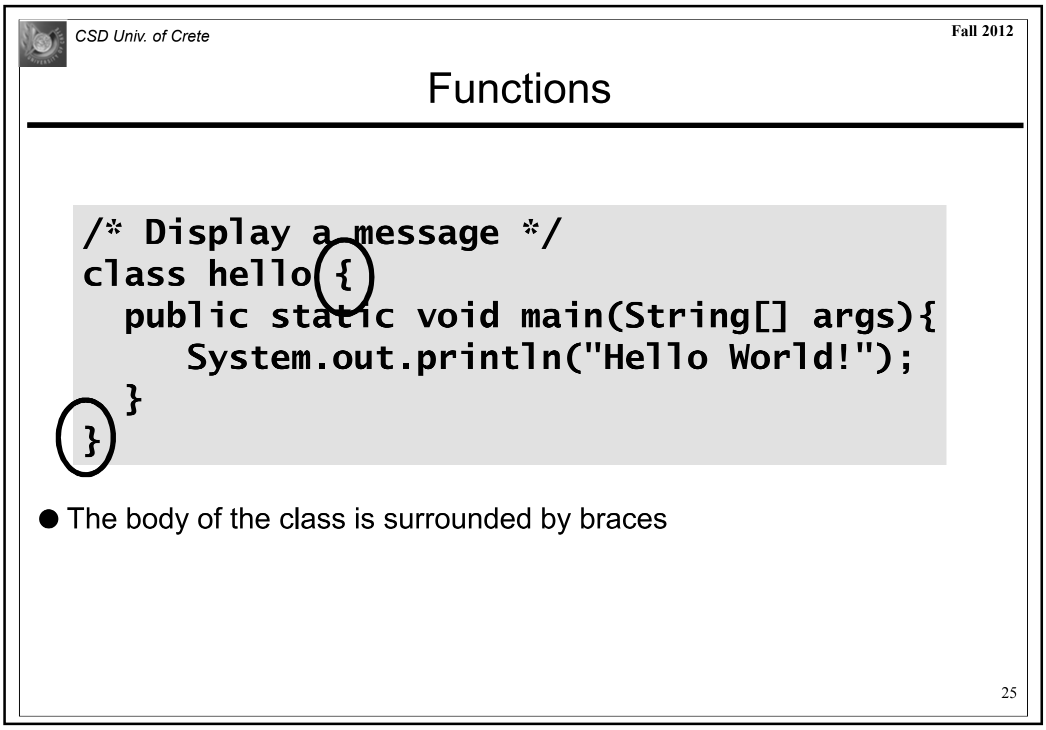
A Program

```
/* Display a message */
class hello {
  public static void main(String[] args){
     System.out.println("Hello World!");
  }
}
```

Functions

```
/* Display a message */
class hello {
  public static void main(String[] args){
     System.out.println("Hello World!");
  }
}
```

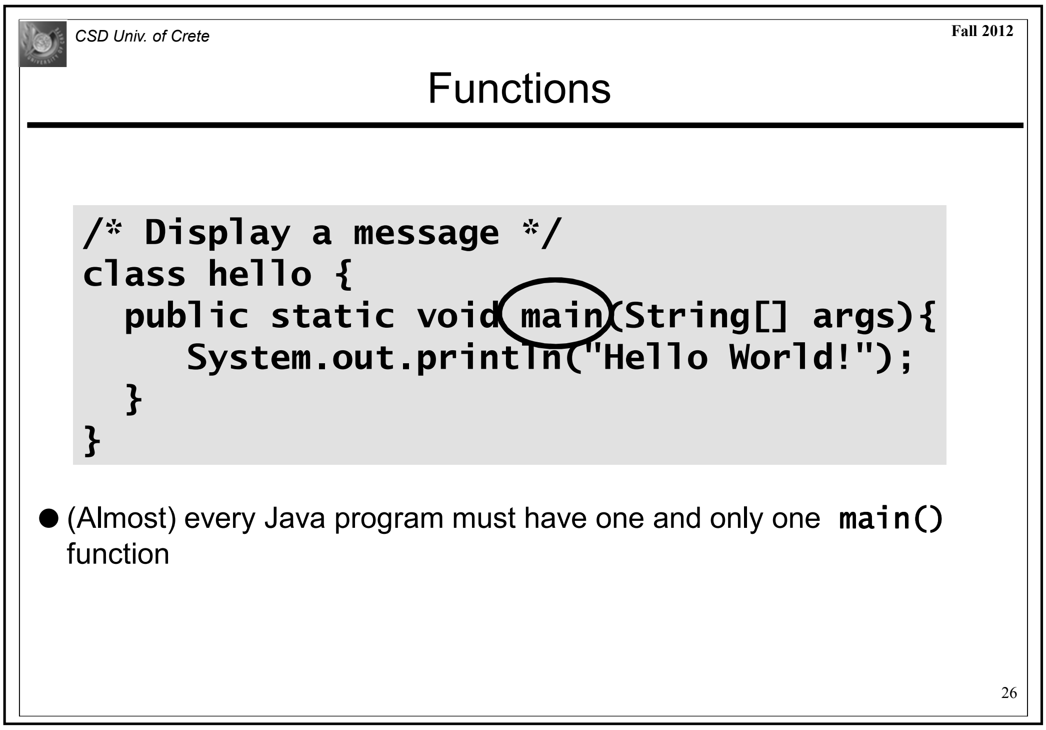
- Java program consists of a named class



Functions

```
/* Display a message */
class hello {
  public static void main(String[] args){
     System.out.println("Hello World!");
  }
}
```

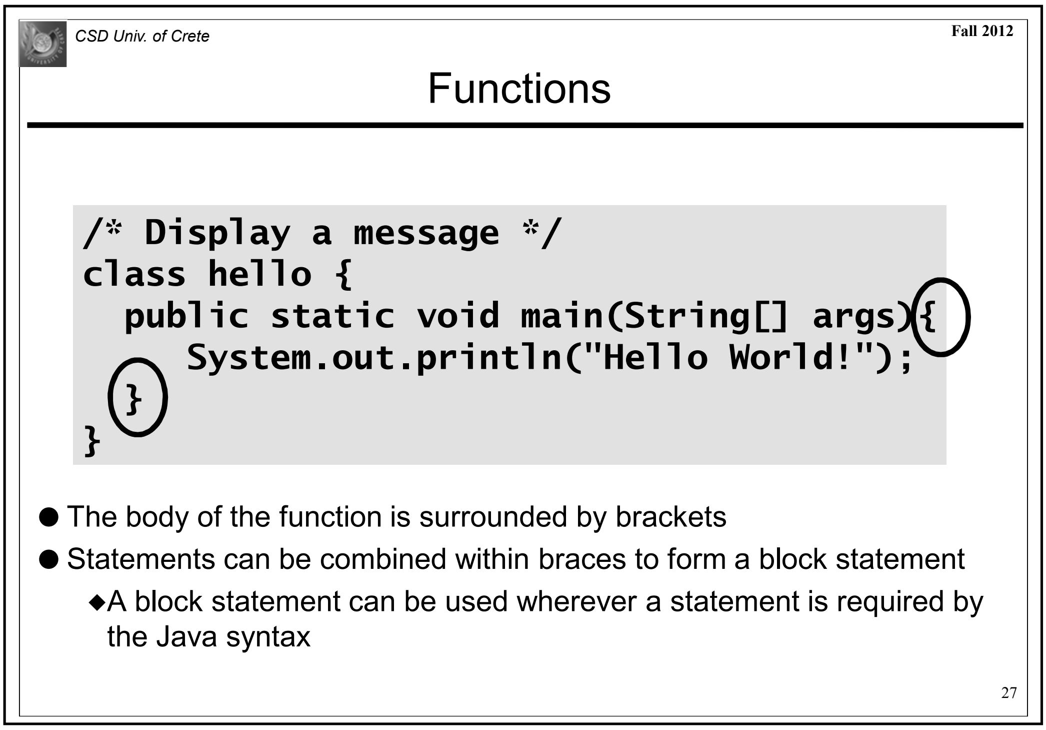
- The body of the class is surrounded by braces



Functions

```
/* Display a message */
class hello {
  public static void main(String[] args){
     System.out.println("Hello World!");
  }
}
```

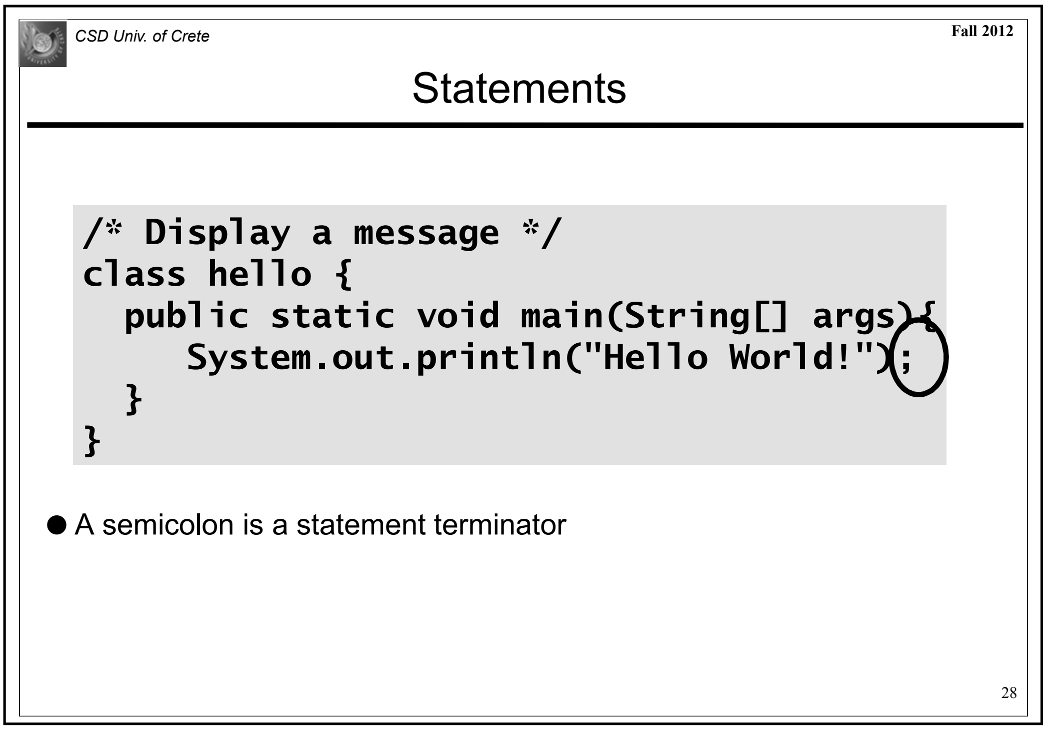
- (Almost) every Java program must have one and only one `main()` function



Functions

```
/* Display a message */
class hello {
  public static void main(String[] args){
     System.out.println("Hello World!");
  }
}
```

- The body of the function is surrounded by brackets
- Statements can be combined within braces to form a block statement
  - A block statement can be used wherever a statement is required by the Java syntax



Statements

```
/* Display a message */
class hello {
  public static void main(String[] args){
     System.out.println("Hello World!");
  }
}
```

- A semicolon is a statement terminator

Java Separators

- Nine ASCII characters are the Java punctuators (separators)

{ }	curly braces	(" code block for a group of statements ")
[ ]	square braces	(" array element size ")
( )	parenthesis	(" groups operations ")
;	semi-colon	(" ends a java statement ")
,	comma	(" inside of a for loop ")
:	colon	(" with a label with break or continue ")

White Spaces

```
/* Display a message */
class hello {
  public static void main(String[] args){
     System.out.println("Hello World!");
  }
}
```

- Spaces, blank lines, and tabs are collectively called white space and are used to separate words and symbols in a program
  - Extra white space is ignored, so a valid Java program can be formatted in many different ways
  - Programs should be formatted to enhance readability, using consistent indentation

Objects

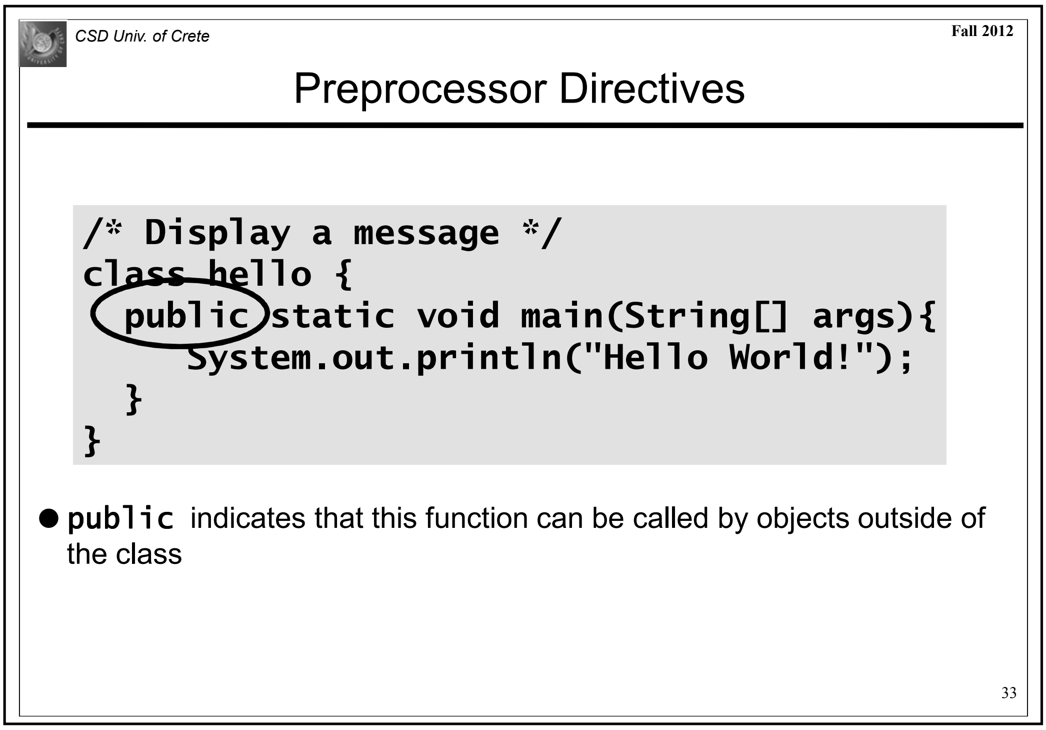
```
/* Display a message */
class hello {
  public static void main(String[] args){
     System.out.println("Hello World!");
  }
}
```

- The identifier `System.out` is an object
- The identifier `println` is one of the methods for that object

Strings

```
/* Display a message */
class hello {
  public static void main(String[] args){
     System.out.println("Hello World!");
  }
}
```

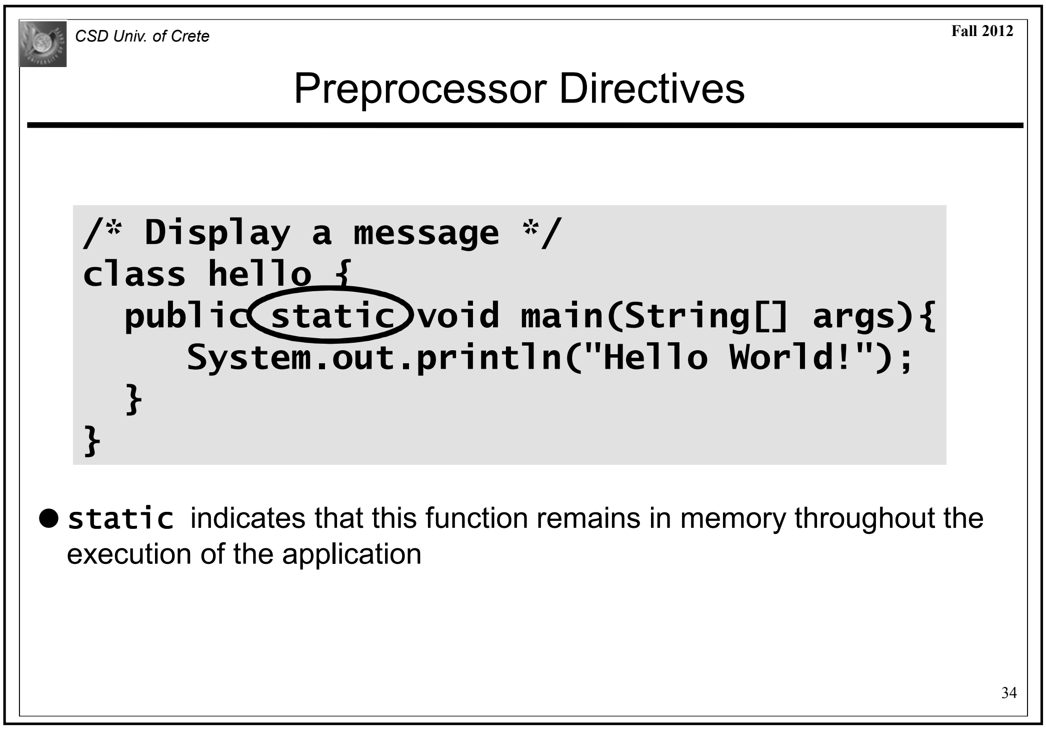
- `"Hello World"` is called an object string
- There is an explicit string type (class) in Java (unlike C/C++)
  - ◆Strings are different than characters !!!



Preprocessor Directives

```
/* Display a message */
class hello {
  public static void main(String[] args){
     System.out.println("Hello World!");
  }
}
```

- `public` indicates that this function can be called by objects outside of the class



Preprocessor Directives

```
/* Display a message */
class hello {
  public static void main(String[] args){
     System.out.println("Hello World!");
  }
}
```

- `static` indicates that this function remains in memory throughout the execution of the application

Preprocessor Directives

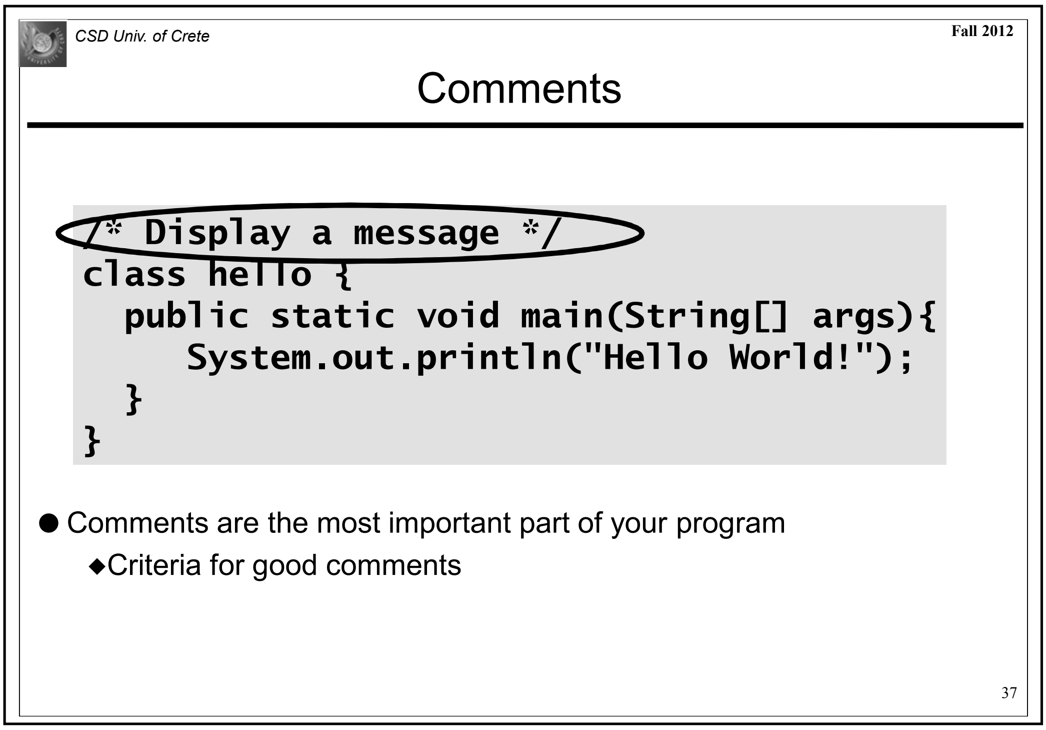
```
/* Display a message */
class hello {
  public static void main(String[] args){
     System.out.println("Hello World!");
  }
}
```

- `void` indicates that this function does not return a value to the object that calls it

Preprocessor Directives

```
/* Display a message */
class hello {
  public static void main(String[] args){
     System.out.println("Hello World!");
  }
}
```

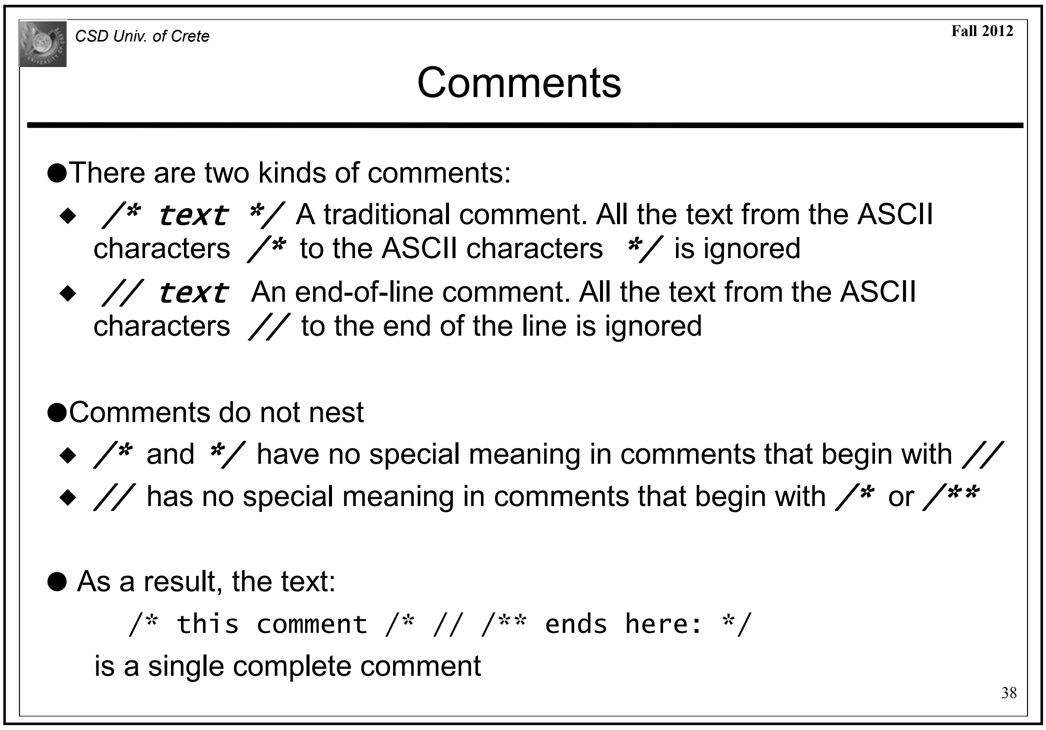
- `args` can be used in the `main` function to pass parameters from the operating system command line



Comments

```
/* Display a message */
class hello {
  public static void main(String[] args){
     System.out.println("Hello World!");
  }
}
```

- Comments are the most important part of your program
  - Criteria for good comments



Comments

- There are two kinds of comments:
  - `/* text */` A traditional comment. All the text from the ASCII characters `/*` to the ASCII characters `*/` is ignored
  - `// text` An end-of-line comment. All the text from the ASCII characters `//` to the end of the line is ignored

- Comments do not nest
  - `/*` and `*/` have no special meaning in comments that begin with `//`
  - `//` has no special meaning in comments that begin with `/*` or `/**`

- As a result, the text:

```
/* this comment /* // /** ends here: */
```

is a single complete comment

Java Identifiers

Java Identifiers

- Identifiers are the words a programmer uses in a program
  - ◆Most identifiers have no predefined meaning except as specified by the programmer
- Rules...
  - ◆An identifier can be made up of letters, digits, the underscore character (_), and the dollar sign (**$**)
  - ◆The first character must be any non-digit from the Unicode standard
  - ◆Subsequent characters may include digits
  - ◆Avoid using underscore and **$** for the first character
  - ◆Java is case sensitive, therefore `Total` and `total` are different identifiers
- Three types of identifiers**:**
  - ◆words that we make up ourselves
  - ◆words that are reserved for special purposes in the language
  - ◆words that are not in the language, but were used by other programmers to make the library

Java Reserved Words

- Some identifiers, called reserved words, have specific meanings in Java and cannot be used in other ways
  - ◆User-defined identifiers cannot duplicate Java reserved words (aka keywords)

abstract
boolean
break
byte
byvalue
case
cast
catch
char
class
const
continue

default
do
double
else
extends
false
final
finally
float
for
future
generic

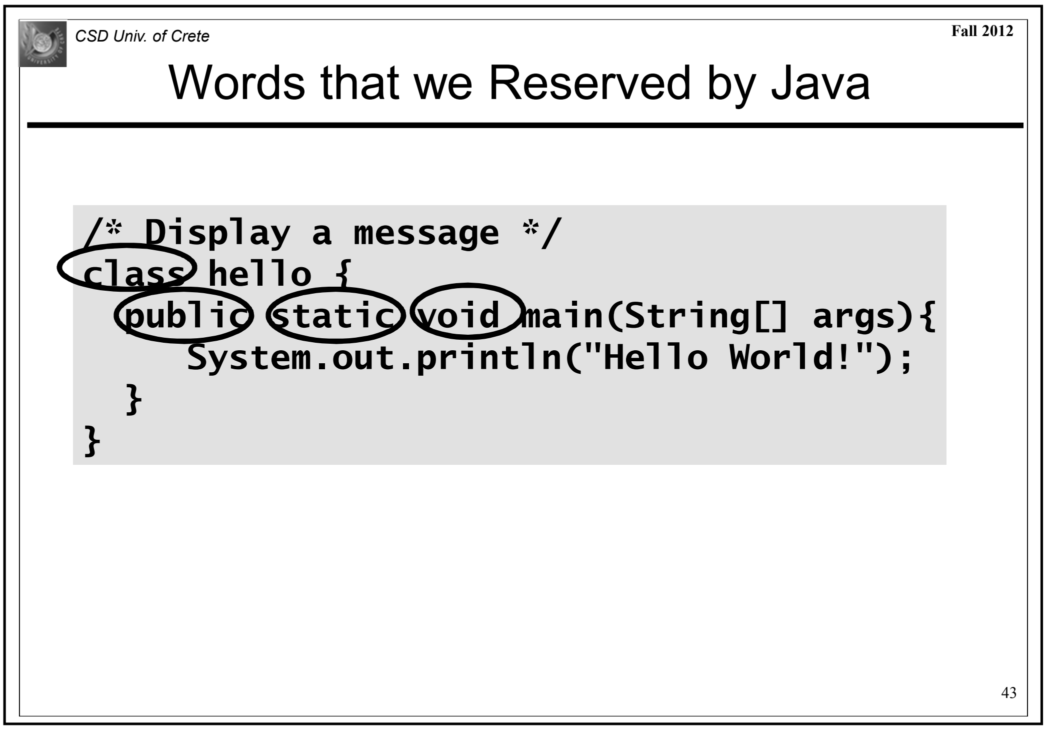
goto
if
implements
import
inner
instanceof
int
interface
long
native
new
null

operator
outer
package
private
protected
public
rest
return
strictfp
short
static
super

switch
synchronized
this
throw
throws
transient
true
try
var
void
volatile
while

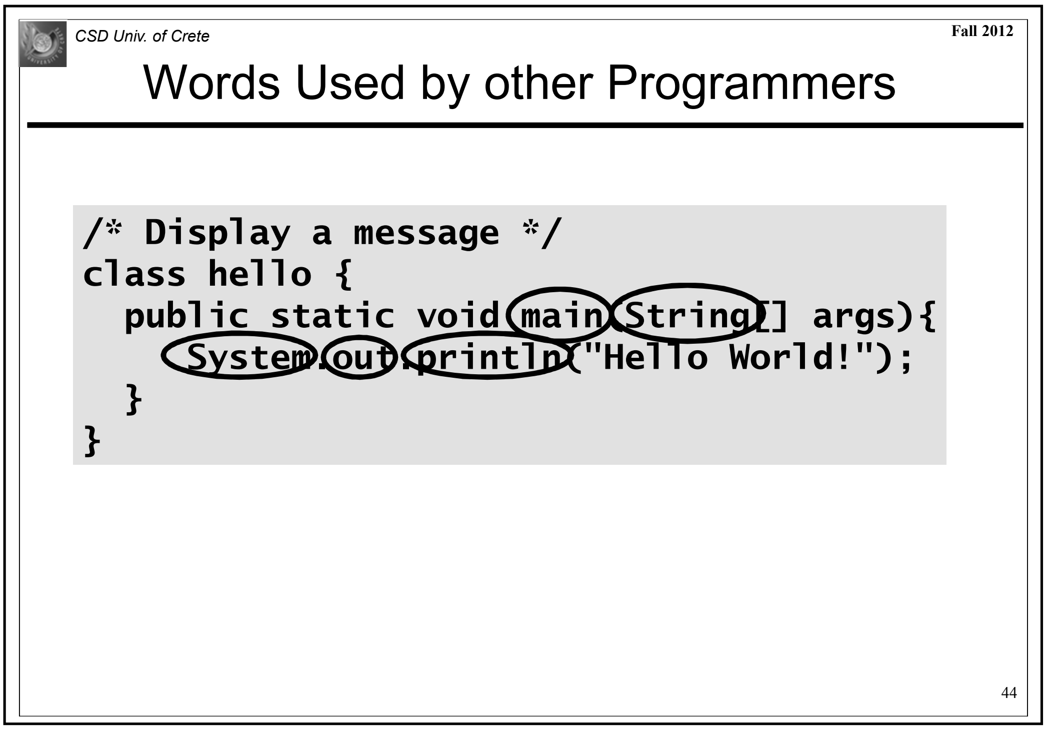
Words that we Make up Ourselves

```
/* Display a message */
class hello {
  public static void main(String[] args){
     System.out.println("Hello World!");
  }
}
```



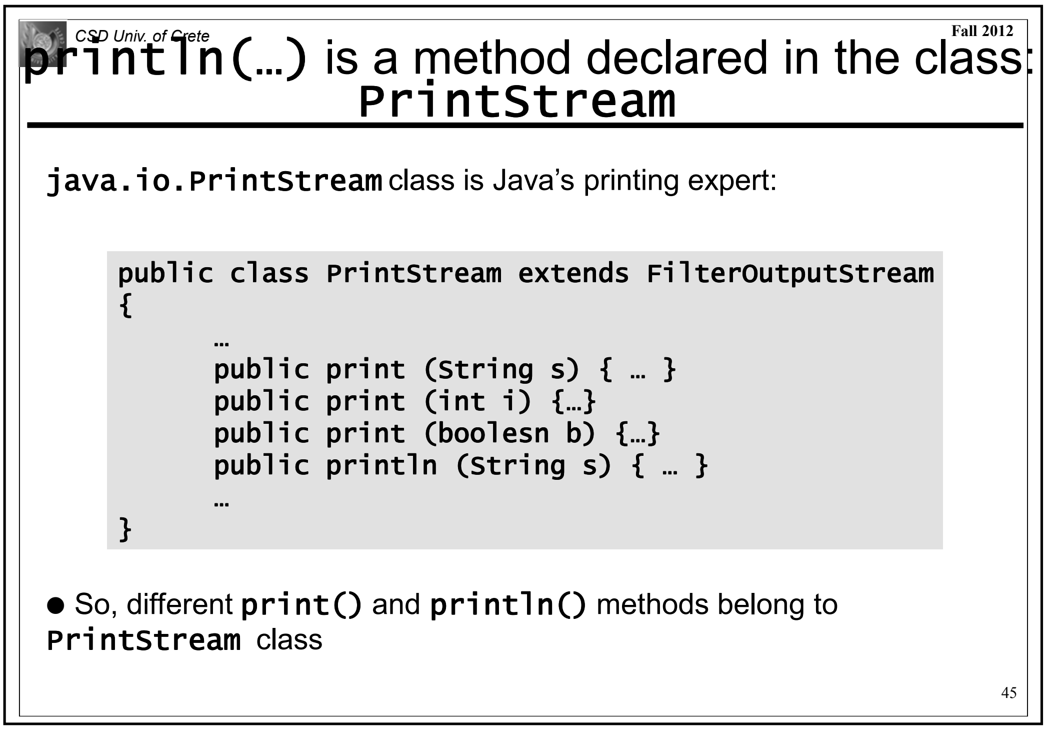
Words that we Reserved by Java

```
/* Display a message */
class hello {
  public static void main(String[] args){
     System.out.println("Hello World!");
  }
}
```



Words Used by other Programmers

```
/* Display a message */
class hello {
  public static void main(String[] args){
     System.out.println("Hello World!");
  }
}
```

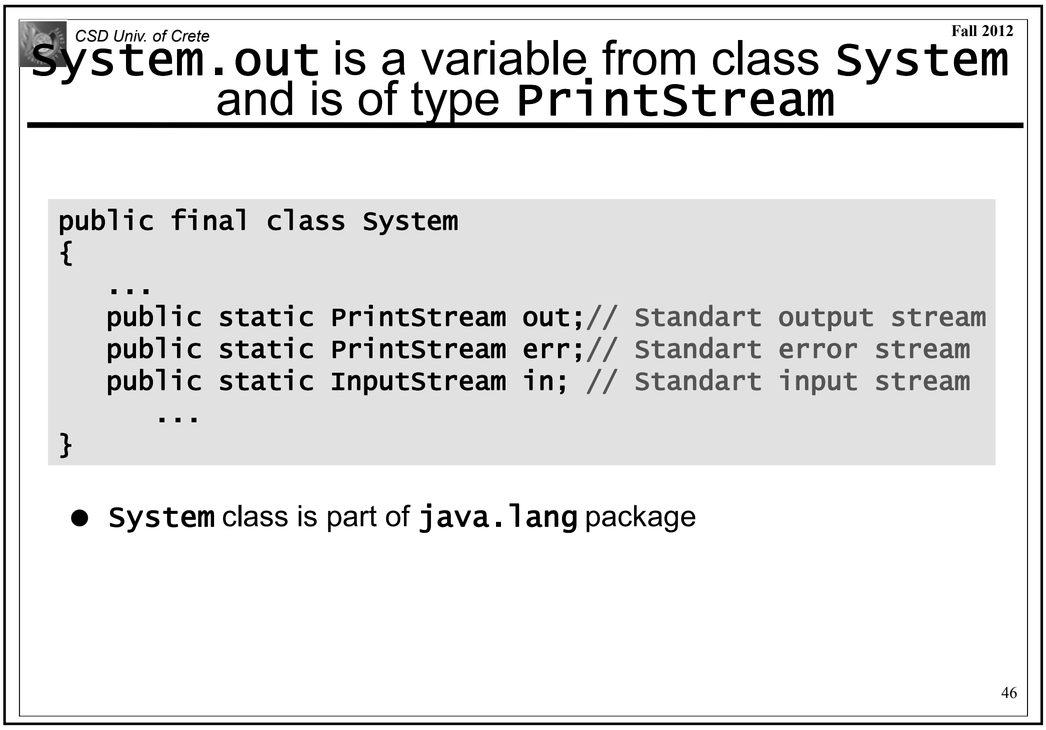


`println(…)` is a method declared in the class: `PrintStream`

`java.io.PrintStream` class is Java’s printing expert:

```
public class PrintStream extends FilterOutputStream
{
      …
      public print (String s) { … }
      public print (int i) {…}
      public print (boolesn b) {…}
      public println (String s) { … }
      …
}
```

- So, different `print()` and `println()` methods belong to `PrintStream` class

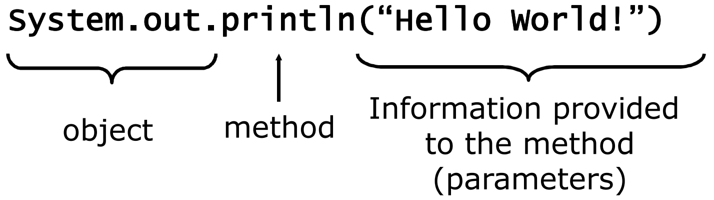


`System.out` is a variable from class `System` and is of type `PrintStream`

```
public final class System
{
   ...
   public static PrintStream out;// Standart output stream
   public static PrintStream err;// Standart error stream
   public static InputStream in; // Standart input stream
      ...
}
```

- `System` class is part of `java.lang` package

Putting them all Together



```
System.out.println("Hello World!")
```

object

method

Information provided to the method (parameters)

- A method `println(…)` is a service that the `System.out` object can perform
  - ◆This is the object of type `PrintStream`, declared in `java.lang.System` class
- Method `println(…)` is invoked (or called)
- The method `println(…)` is called by sending the message to the `System.out` object, requesting the service

Naming Style of Identifiers

- Names should be chosen carefully: they play a central role in the readability of the program and is part of its documentation; they should be:
  - ◆meaningful

    `BankAccount, size` vs. `XP12_r$, wq1`

  - ◆long enough to express the meaning of the name

    `numberOfElements`

  - ◆But not unnecessarily long

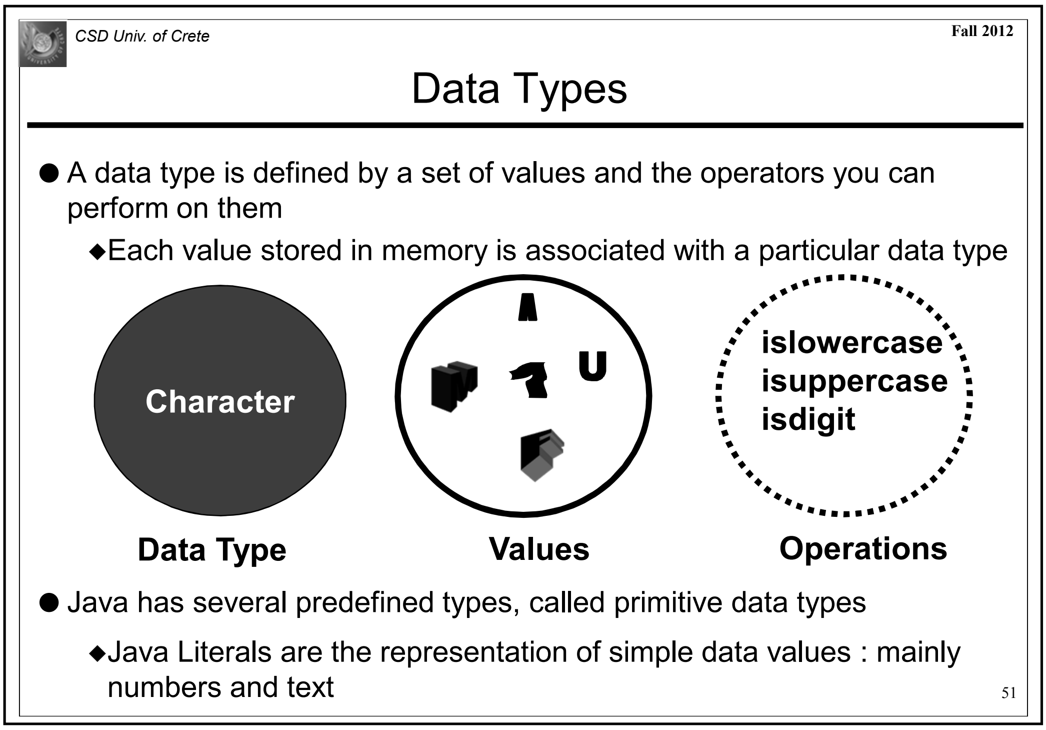
    `theCurrentItemBeingProcessed`

All Identifiers have an Associated Scope

- The scope of a name is the region of program code where that name is visible
- A name cannot be accessed outside its scope
- Within a method:
  - Braces (`{}`) mark closed regions of program statements
  - A local variable is only visible from the point of its declaration until the closing brace enclosing it

```
Public void thisMethod() {
    int variable1 = ...;
    while ( notDone ) {
        int variable2 = ...;
     }
}
```

Java Types and Variables



Data Types

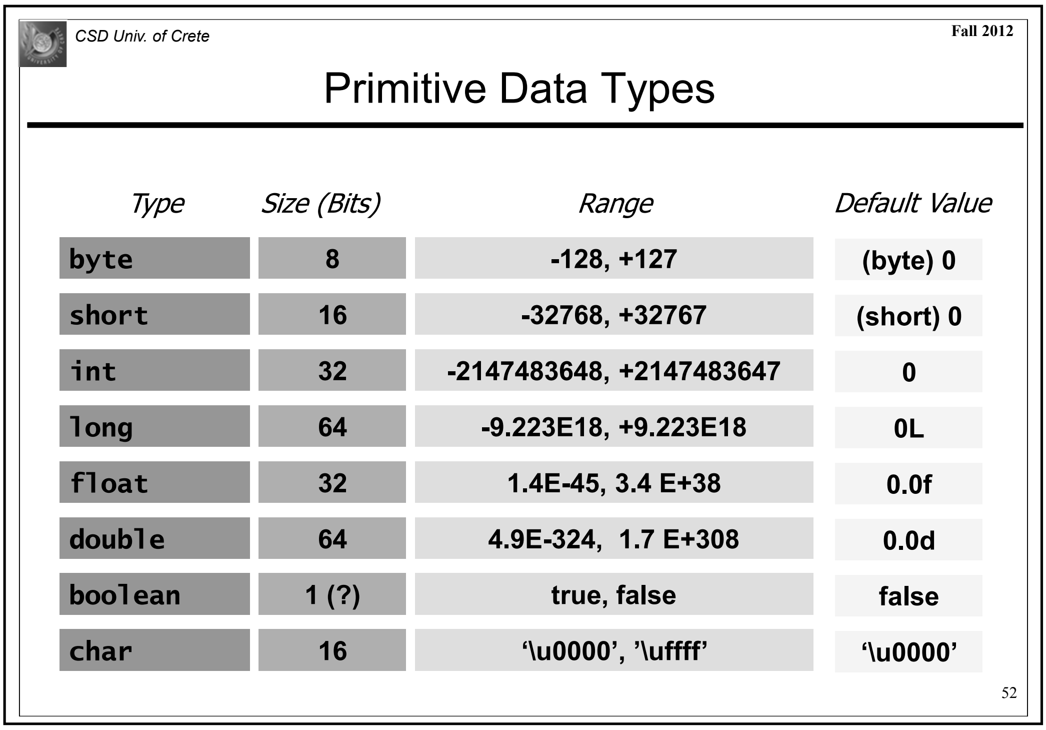
- A data type is defined by a set of values and the operators you can perform on them
  - ◆Each value stored in memory is associated with a particular data type

Character

islowercase
isuppercase
isdigit

Data Type Values Operations

- Java has several predefined types, called primitive data types
  - ◆Java Literals are the representation of simple data values : mainly numbers and text



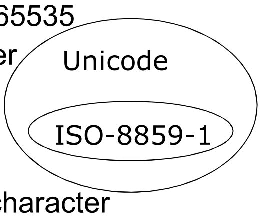
Primitive Data Types

Type	*Size (Bits)*	*Range*	*Default Value*
`byte`	8	-128, +127	(byte) 0
`short`	16	-32768, +32767	(short) 0
`int`	32	-2147483648, +2147483647	0
`long`	64	-9.223E18, +9.223E18	0L
`float`	32	1.4E-45, 3.4 E+38	0.0f
`double`	64	4.9E-324, 1.7 E+308	0.0d
`boolean`	1 (?)	true, false	false
`char`	16	‘\u0000’, ’\uffff’	‘\u0000’

Unicode Character Type

● A `char` value stores a single character from the Unicode character set

- ◆ The total number of represented characters is 2^{16} = 65535
- ◆ The Unicode character set uses 16 bits per character
- ◆ Check out codes at http://unicode.org



● The ASCII character set is still the basis for many other programming languages

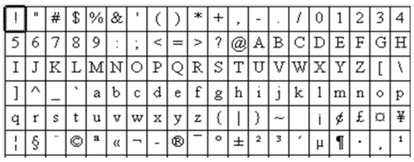
- ◆ The ASCII character set uses 8 bits (one byte) per character
- ◆ To provide backwards compatibility with the ASCII code, the first 128 characters are the ASCII coded characters

● Java 5 supports Unicode 4.0, which defines some characters that require 21 bits in addition to the 16-bit standard Unicode 3.0 characters

- ◆ You use int to represent these characters, and some of the static methods in the Character class now accept int arguments to deal with them.
- ◆ In a String, use a pair of char values to encode a 21-bit character

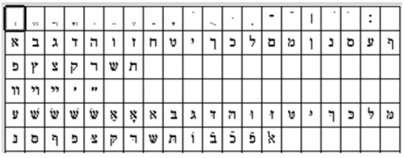
Part of the Unicode Set

0x0021 …



!	"	#	$	%	&	'	(	)	*	+	,	-	.	/	0	1	2	3	4
5	6	7	8	9	:	;	<	=	>	?	@	A	B	C	D	E	F	G	H
I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	[	\
]	^	_	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p
q	r	s	t	u	v	w	x	y	z	{	|	}	~		¡	¢	£	¤	¥
¦	§	¨	©	ª	«	¬	-	®	¯	°	±	²	³	´	µ	¶	·	¸	¹

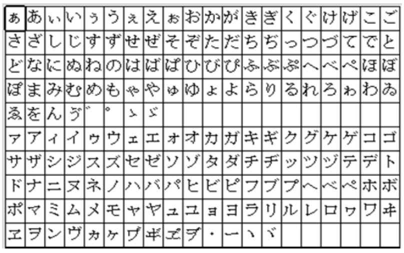
0x05B0 …



ְ	ֱ	ֲ	ֳ	ִ	ֵ	ֶ	ַ	ָ	ֹ	ֻ	ּ	ֽ	־	ֿ	׀	ׁ	ׂ	׃	
א	ב	ג	ד	ה	ו	ז	ח	ט	י	ך	כ	ל	ם	מ	ן	נ	ס	ע	ף
פ	ץ	צ	ק	ר	ש	ת													
װ	ױ	ײ	׳	״															
ﬠ	שׁ	שׂ	שּׁ	שּׂ	אַ	אָ	אּ	בּ	גּ	דּ	הּ	וּ	זּ	טּ	יּ	ךּ	כּ	לּ	מּ
נּ	סּ	ףּ	פּ	צּ	קּ	רּ	שּ	תּ	וֹ	בֿ	כֿ	פֿ	ﭏ						

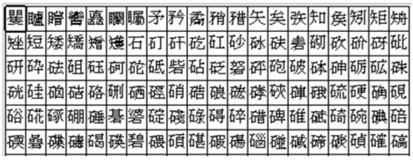
…

0x3041 …



ぁ	あ	ぃ	い	ぅ	う	ぇ	え	ぉ	お	か	が	き	ぎ	く	ぐ	け	げ	こ	ご
さ	ざ	し	じ	す	ず	せ	ぜ	そ	ぞ	た	だ	ち	ぢ	っ	つ	づ	て	で	と
ど	な	に	ぬ	ね	の	は	ば	ぱ	ひ	び	ぴ	ふ	ぶ	ぷ	へ	べ	ぺ	ほ	ぼ
ぽ	ま	み	む	め	も	ゃ	や	ゅ	ゆ	ょ	よ	ら	り	る	れ	ろ	ゎ	わ	ゐ
ゑ	を	ん	ゔ	゛	゜	ゝ	ゞ												
ァ	ア	ィ	イ	ゥ	ウ	ェ	エ	ォ	オ	カ	ガ	キ	ギ	ク	グ	ケ	ゲ	コ	ゴ
サ	ザ	シ	ジ	ス	ズ	セ	ゼ	ソ	ゾ	タ	ダ	チ	ヂ	ッ	ツ	ヅ	テ	デ	ト
ド	ナ	ニ	ヌ	ネ	ノ	ハ	バ	パ	ヒ	ビ	ピ	フ	ブ	プ	ヘ	ベ	ペ	ホ	ボ
ポ	マ	ミ	ム	メ	モ	ャ	ヤ	ュ	ユ	ョ	ヨ	ラ	リ	ル	レ	ロ	ヮ	ワ	ヰ
ヱ	ヲ	ン	ヴ	ヵ	ヶ	ヷ	ヸ	ヹ	ヺ	・	ー	ヽ	ヾ						

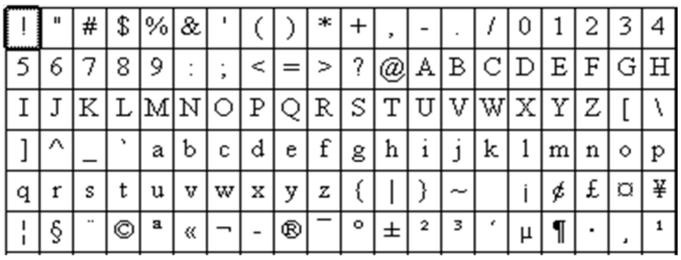
0x77CD …



矍	矑	矒	矕	矗	矙	矚	矛	矜	矞	矟	矠	矢	矣	矤	知	矦	矧	矩	矪
矬	短	矮	矯	矰	矱	石	矴	矸	矻	矼	砂	砅	砆	砉	砌	砍	砎	砑	砒
研	砕	砝	砠	砡	砢	砣	砥	砦	砧	砭	砮	砰	砲	破	砵	砷	砺	砿	硃
硄	硅	硇	硈	硌	硏	硒	硜	硝	硞	硠	硡	硣	硤	硨	硪	硫	硬	确	硯
硲	硴	硺	硼	硾	碁	碆	碇	碊	碌	碍	碎	碏	碑	碓	碔	碕	碗	碘	碚
碝	碞	碟	碡	碣	碤	碧	碨	碩	碪	碬	碭	碯	碰	碱	碲	碳	碵	確	碻

Part of the ASCII Set

0x21 …



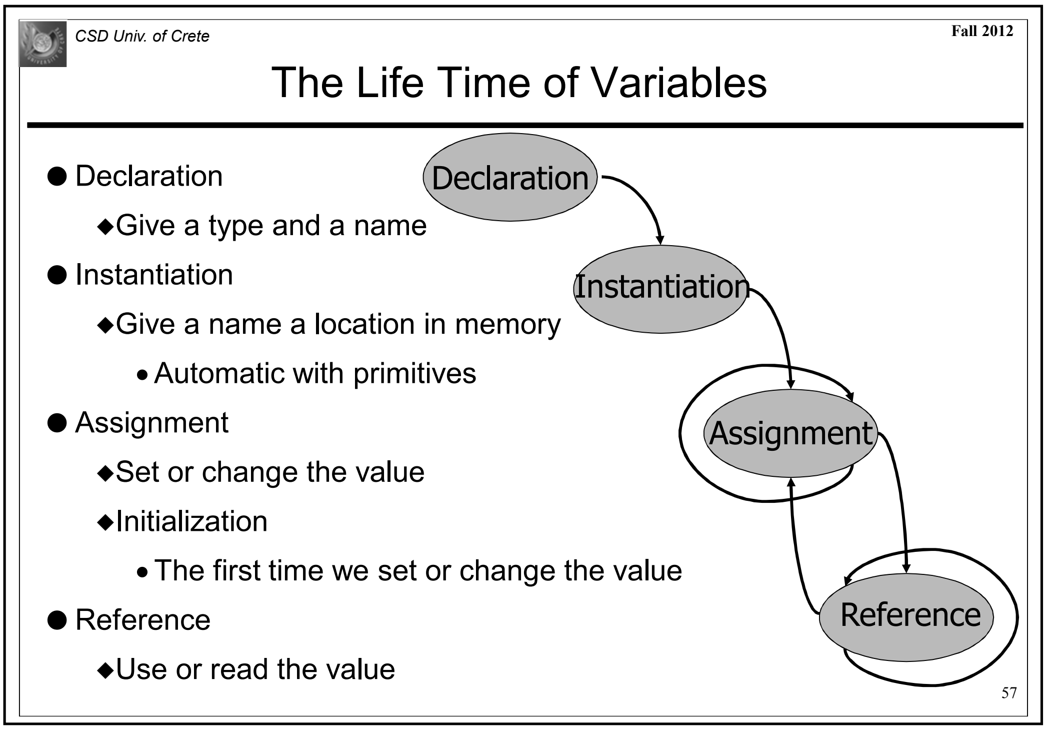
!	"	#	$	%	&	'	(	)	*	+	,	-	.	/	0	1	2	3	4
5	6	7	8	9	:	;	<	=	>	?	@	A	B	C	D	E	F	G	H
I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	[	\
]	^	_	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p
q	r	s	t	u	v	w	x	y	z	{	|	}	~		¡	¢	£	¤	¥
¦	§	¨	©	ª	«	¬	-	®	¯	°	±	²	³	´	µ	¶	·	¸	¹

…0xB9

The Great Escape

- What is an escape sequence?
  - Method for writing “difficult to represent” characters
  - Include invisible and punctuation characters
- Parts of an escape sequence
  - The escape character: the backslash (\)
  - The translation value
    - Unicode values: `‘\u2122’` (Hex -decimal value)
    - When using an octal number you may use 1, 2, or 3 digits
      - Each digit must be in the range 0-7

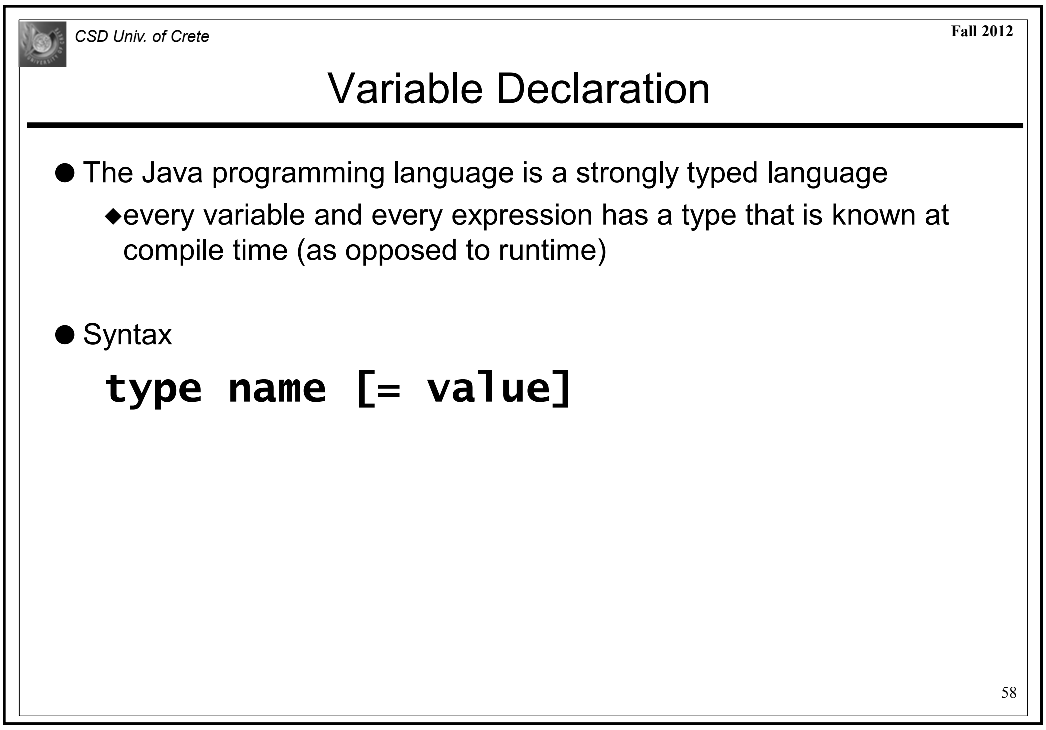
● `\b`	Backspace
● `\t`	Tab
● `\n`	New line
● `\f`	Form feed
● `\r`	Return (ENTER)
● `\”`	Double quote
● `\’`	Single quote
● `\\`	Backslash
● `\ddd`	Octal code
● `\uXXXX`	Hex-decimal code



The Life Time of Variables

- Declaration
  - ◆Give a type and a name
- Instantiation
  - ◆Give a name a location in memory
    - Automatic with primitives
- Assignment
  - ◆Set or change the value
  - ◆Initialization
    - The first time we set or change the value
- Reference
  - ◆Use or read the value

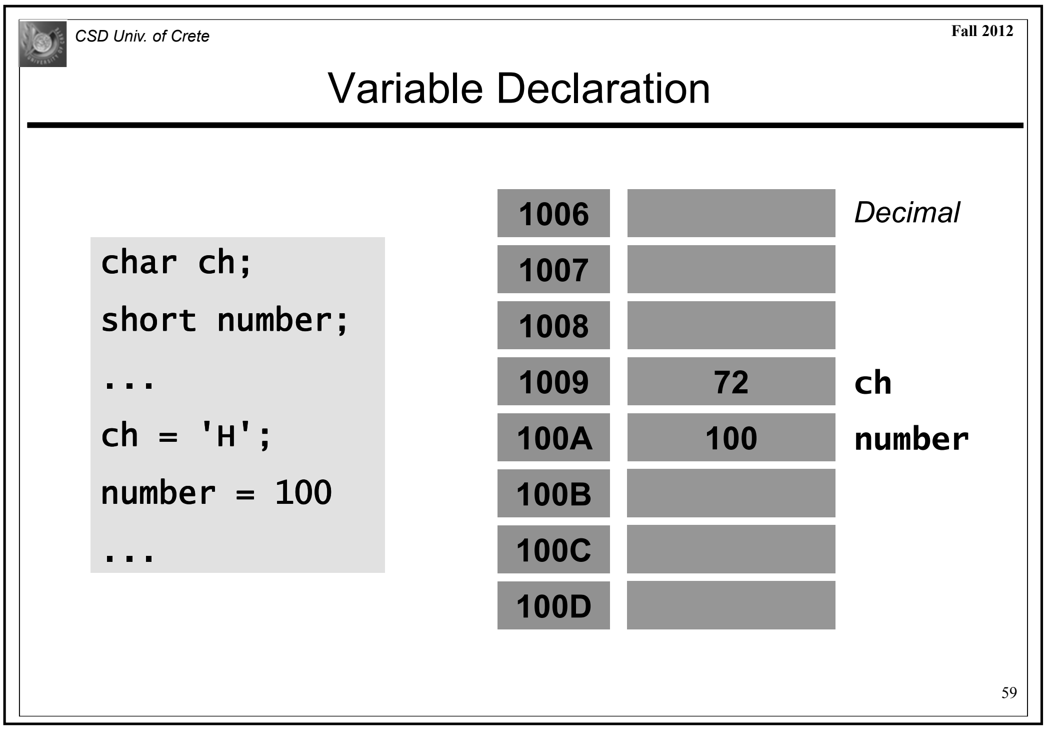
Declaration
Instantiation
Assignment
Reference



Variable Declaration

- The Java programming language is a strongly typed language
  - ◆every variable and every expression has a type that is known at compile time (as opposed to runtime)
- Syntax

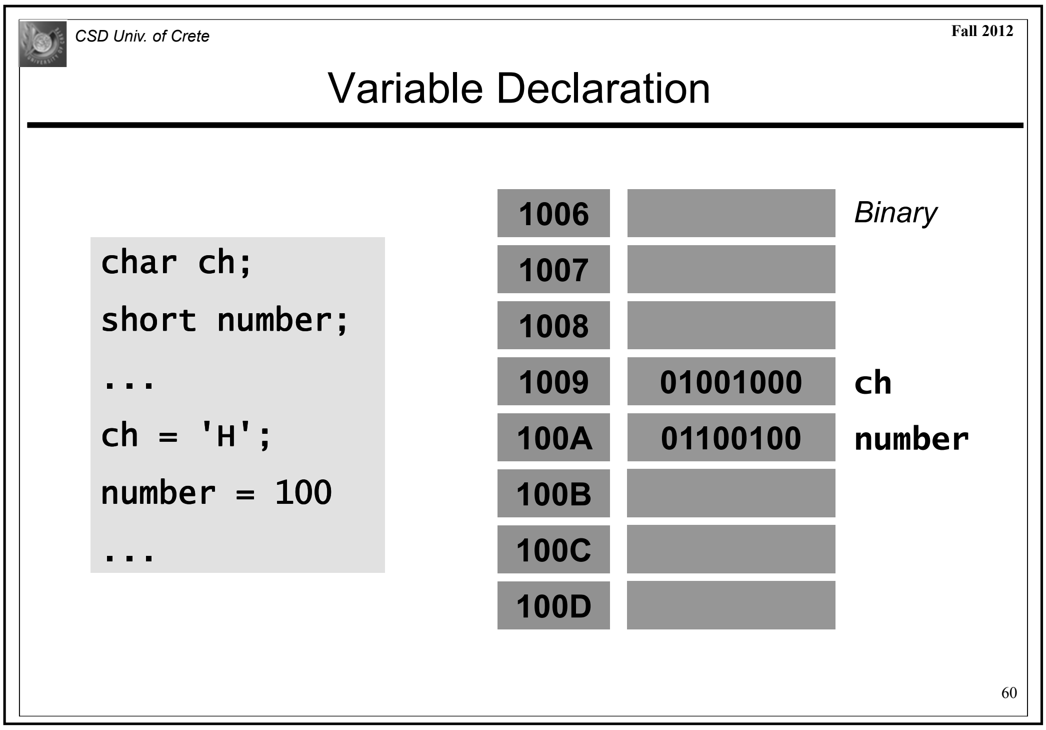
```
type name [= value]
```



Variable Declaration

```
char ch;
short number;
...
ch = 'H';
number = 100
...
```

1006		*Decimal*
1007		
1008		
1009	72	**ch**
100A	100	**number**
100B		
100C		
100D		



Variable Declaration

```
char ch;
short number;
...
ch = 'H';
number = 100
...
```

1006		*Binary*
1007		
1008		
1009	01001000	**ch**
100A	01100100	**number**
100B		
100C		
100D		

Variable Declaration

```
char ch;
short number;
...
ch = 'H';
number = 100
...
```

1006		*Hex*
1007		
1008		
1009	48	**ch**
100A	64	**number**
100B		
100C		
100D		

Java Data Conversions

- Sometimes it is convenient to convert data from one type to another
  - For example, we may want to treat an integer as a floating point value during a computation
- What are compatible conversions?
  - Data conversions between different types where no information is lost
  - Possible from less precise to more precise types
- Two kinds of data conversions
  - Widening are safest because they tend to go from a small data type to a larger one: char, short ➜ int, long, float, double
  - Narrowing can lose information because they tend to go from a large data type to a smaller one : double, float, long, int ➜ char, short
- In Java, data conversions can occur in three ways:
  - assignment conversion
  - arithmetic promotion
  - casting

Assignment Conversions

- Assignment conversion occurs when a value of one type is assigned to a variable of another
  - ◆Only widening conversions can happen via assignment

```
double d = 123.45F;// OK
float f  = 123.45; // Not OK
byte b   = 128;    // Not OK
long l   = 123;    // OK
```

Floating point constants are assumed to be double, by default!

Arithmetic Promotions

- Arithmetic promotion happens automatically when operators in expressions convert their operands
  - ◆When an integer and a floating-point number are used as operands to a single arithmetic operation, the result is floating point
  - ◆The integer is implicitly converted to a floating-point number before the operation takes place

```
int i=37;
double x=27.475;
System.out.println(“i+x=“+(i+x));
```

Output:

```
i+x=64.475
```

Type Promotion Rules

double	None
float	double
long	float, double
int	long, float, double
char	int, long, float, double
short	int, long, float, double
byte	short, int, long, float, double
boolean	None

Long Integer to Floating point Conversion may result in Precision Loss

```
/*
  Testing data conversion
*/
public class DataConv
{
 public static void main(String [] args)
 {long i=100000001; // 9 digits
  float x;
  x=i;
  System.out.println("x="+x);}
}
```

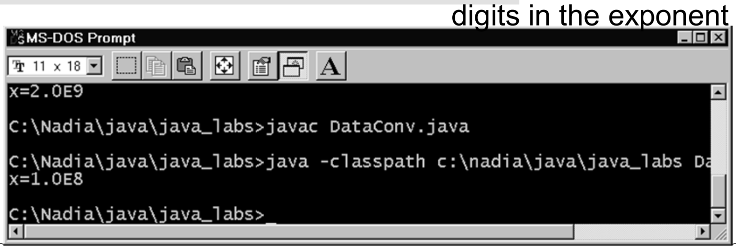
integer values can have up to 10 digits precision

mantissa of a float has only 7-digit precision (23 bits-fraction, 8-exponent, 1-sign sign*mantissa*fraction^{exponent})

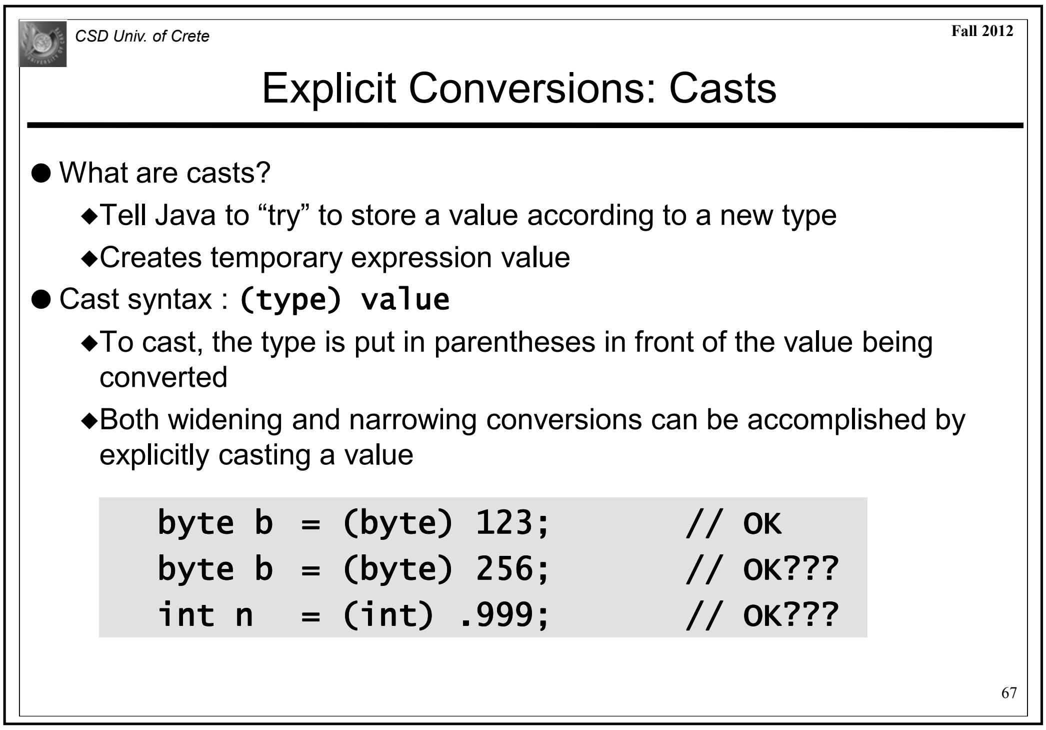
precision:governed by number of digits in the mantissa

range:governed by number of digits in the exponent

Output:



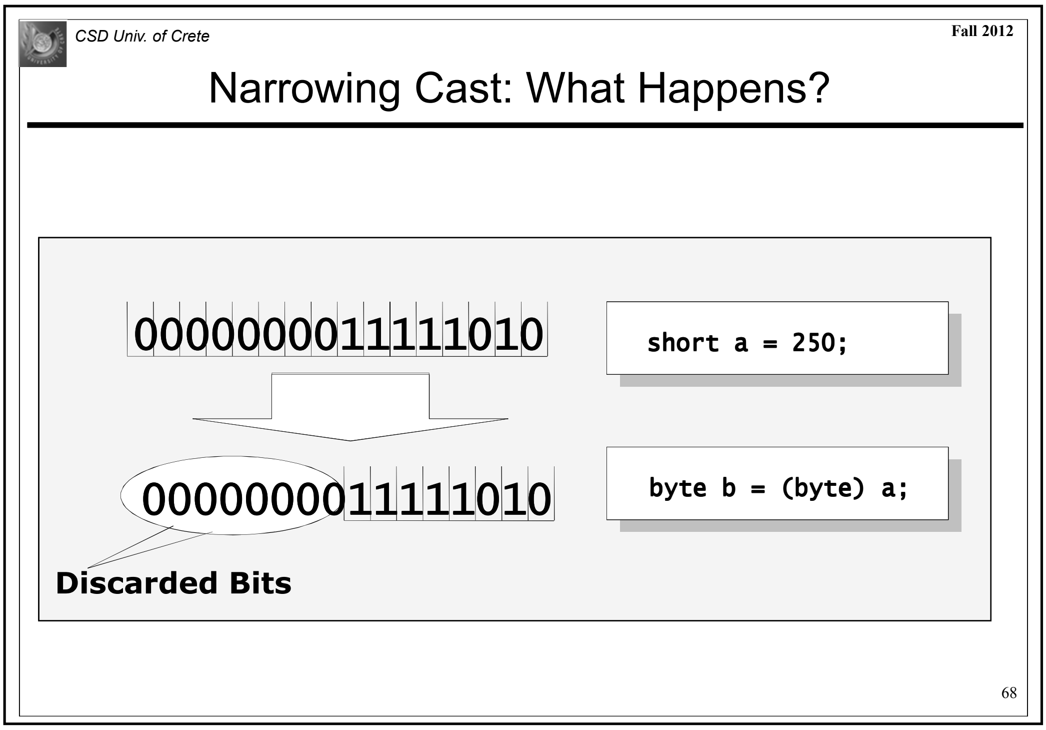
```
x=2.0E9

C:\Nadia\java\java_labs>javac DataConv.java

C:\Nadia\java\java_labs>java -classpath c:\nadia\java\java_labs Da
x=1.0E8

C:\Nadia\java\java_labs>_
```



Explicit Conversions: Casts

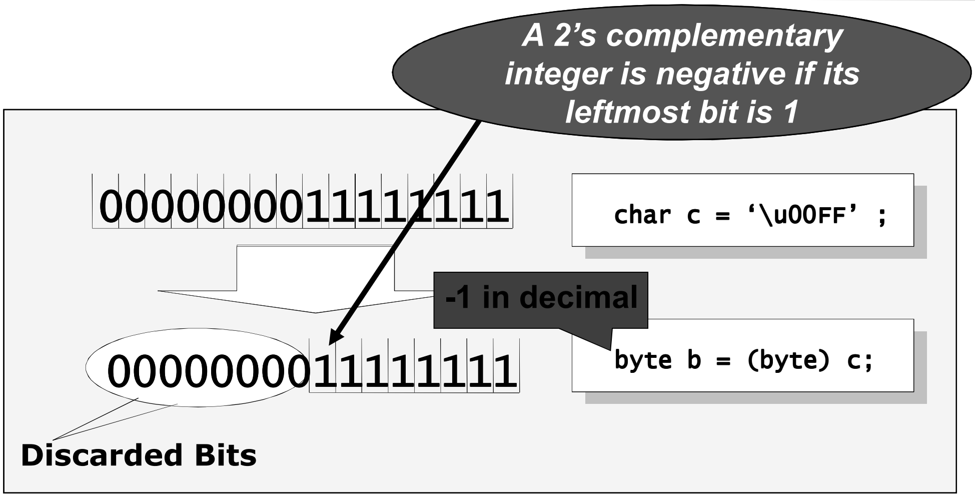
- What are casts?
  - ◆Tell Java to “try” to store a value according to a new type
  - ◆Creates temporary expression value
- Cast syntax : **(type) value**
  - ◆To cast, the type is put in parentheses in front of the value being converted
  - ◆Both widening and narrowing conversions can be accomplished by explicitly casting a value

```
byte b  = (byte) 123;       // OK
byte b  = (byte) 256;       // OK???
int n   = (int) .999;       // OK???
```



Narrowing Cast: What Happens?

Narrowing Cast: What Happens?



A 2's complementary integer is negative if its leftmost bit is 1

0000000011111111

char c = '\u00FF' ;

-1 in decimal

0000000011111111

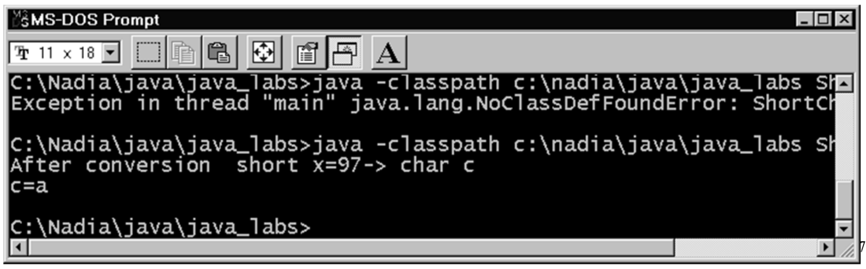
byte b = (byte) c;

Discarded Bits

Narrowing Casts can Lose both Numeric Magnitude and Precision

```
/* short to char conversion   */
public class ShortChar
{
 public static void main(String []args)
  {short x=97;
   char c=(char)x;
   System.out.println("After conversion short x=97->char c");
   System.out.println("c="+c);}
}
```

Output:

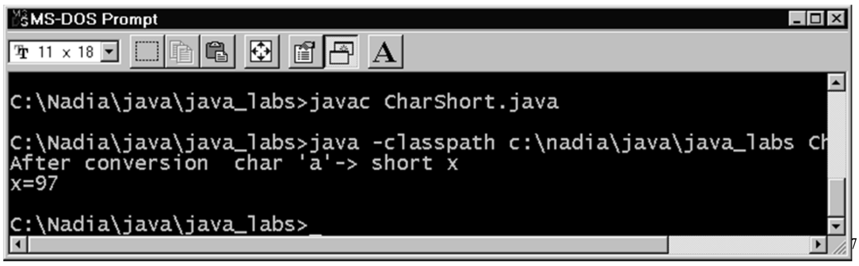


```
C:\Nadia\java\java_labs>java -classpath c:\nadia\java\java_labs Sh
Exception in thread "main" java.lang.NoClassDefFoundError: ShortCh

C:\Nadia\java\java_labs>java -classpath c:\nadia\java\java_labs Sh
After conversion  short x=97-> char c
c=a

C:\Nadia\java\java_labs>
```

Narrowing Casts can Lose both Numeric Magnitude and Precision

```
/*  char to short conversion  */
public class CharShort
{
 public static void main(String []args)
 {char c='a';
  short x=(short)c;
  System.out.println("After conversion char'a'->short x");
  System.out.println("x="+x);}
}
```

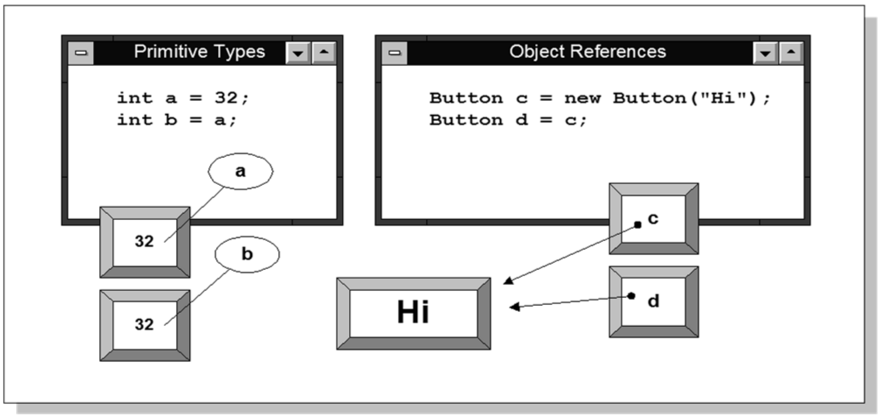
Output:



```
C:\Nadia\java\java_labs>javac CharShort.java

C:\Nadia\java\java_labs>java -classpath c:\nadia\java\java_labs Ch
After conversion  char 'a'-> short x
x=97

C:\Nadia\java\java_labs>_
```

Values and References

- The differences between literals and objects
  - ◆Primitive types hold values
  - ◆Object types (including Strings) hold references



```
int a = 32;
int b = a;
```

```
Button c = new Button("Hi");
Button d = c;
```

36

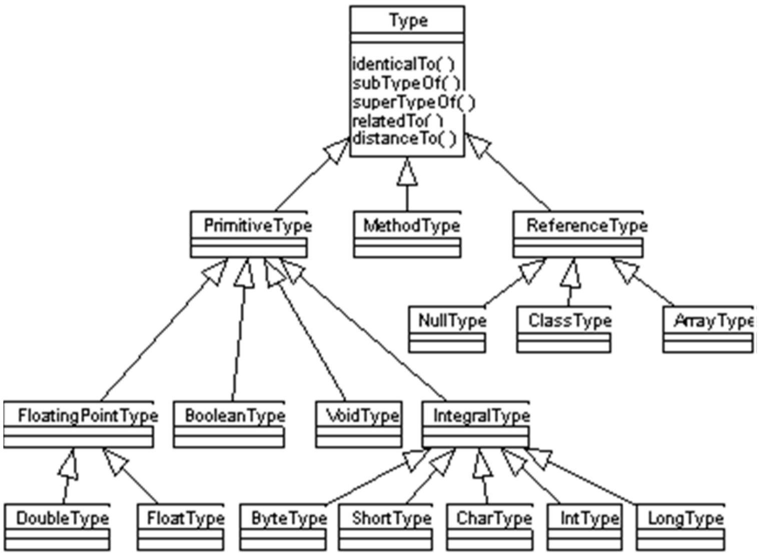
The Null Type

- `Null` type has no name and does not belong to any category
  - It is impossible to declare a variable of the null type

- The null type has one value, the null reference, represented by the literal `null`, which is formed from ASCII characters

- Usually the `null` type is ignored and we pretend that `null` is merely a special literal that can be of any reference type

The Java Typing System

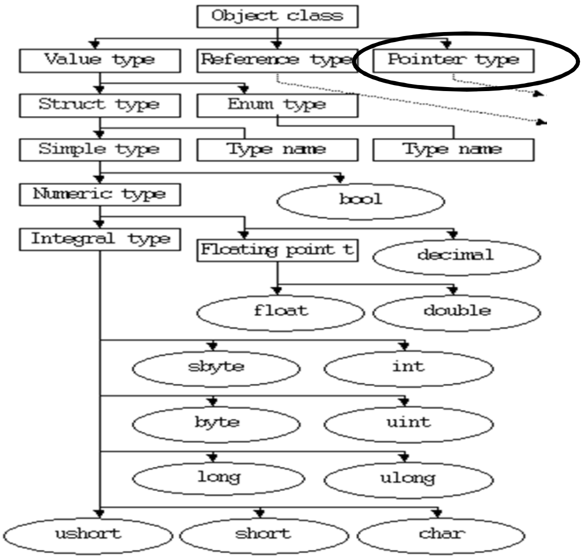


Java vs. C or C++ Data Types

- ● Two type categories
- ● All nonprimitive types are objects
- ● All numeric types are signed
- ● All primitive types are a fixed size for all platforms
- ● 16-bit Unicode characters
- ● Boolean data type primitive
- ● Conditions must be boolean expressions
- ● Variables are automatically initialised

- ● Various type categories
- ● Separate types for structs, unions, enums, and arrays
- ● Signed and unsigned numeric types
- ● Primitive type size varies by platform
- ● 8-bit ASCII characters
- ● No explicit boolean data type
- ● Integer results are interpreted as boolean conditions
- ● No automatic initialisation of variables

Java vs. C++ Typing System



Object class

Value type

Reference type

Pointer type

Struct type

Enum type

Simple type

Type name

Type name

Numeric type

bool

Integral type

Floating point t

decimal

float

double

sbyte

int

byte

uint

long

ulong

ushort

short

char

Java Operators

What’s an Operator?

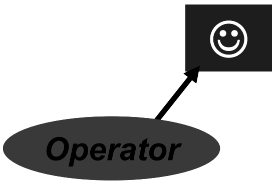
- Operators are tokens that trigger some computation when applied to variables and other objects
  - ◆Arithmetic, logical, and bit-level operators
  - ◆Class access operators
- The Java operators are formed from ASCII characters:

()
++
--
!
~
instance of
*** **
/
%
+
-
<<
>>
>>>
<
>
<=
>=
==
!=
&
^
|
&&
||
?:
=
op=

Unary Operators

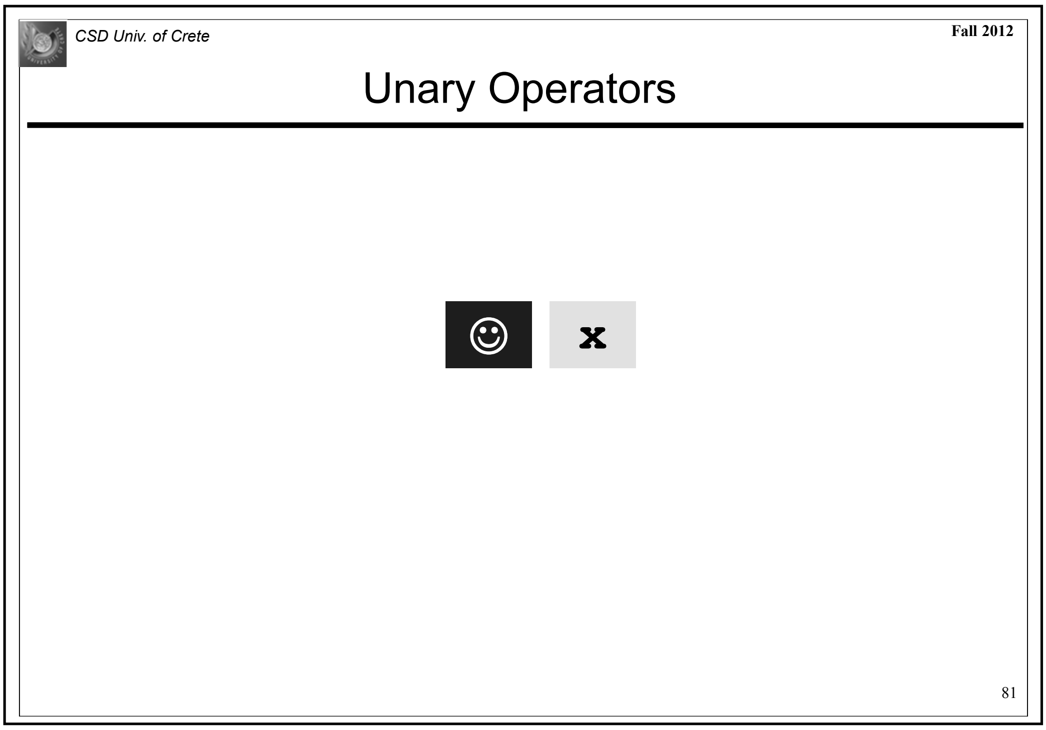
☺

Unary Operators



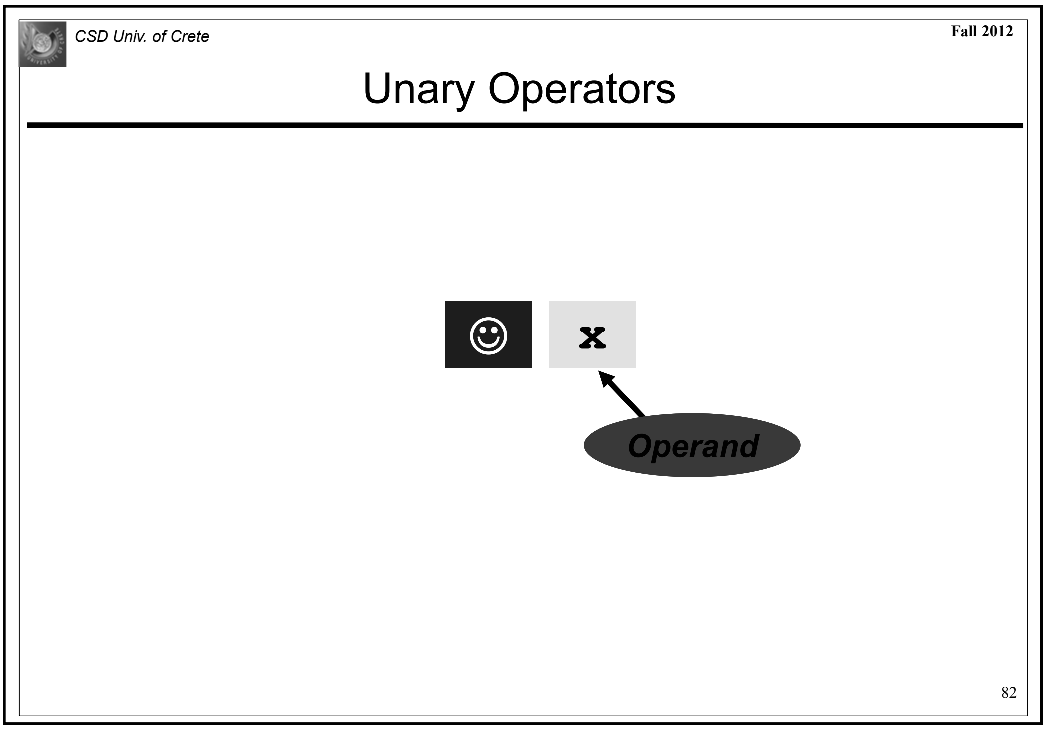
☺

Operator



Unary Operators

☺ x



Unary Operators

☺ x

Operand

Unary Operators

☺ **x**

x ☺

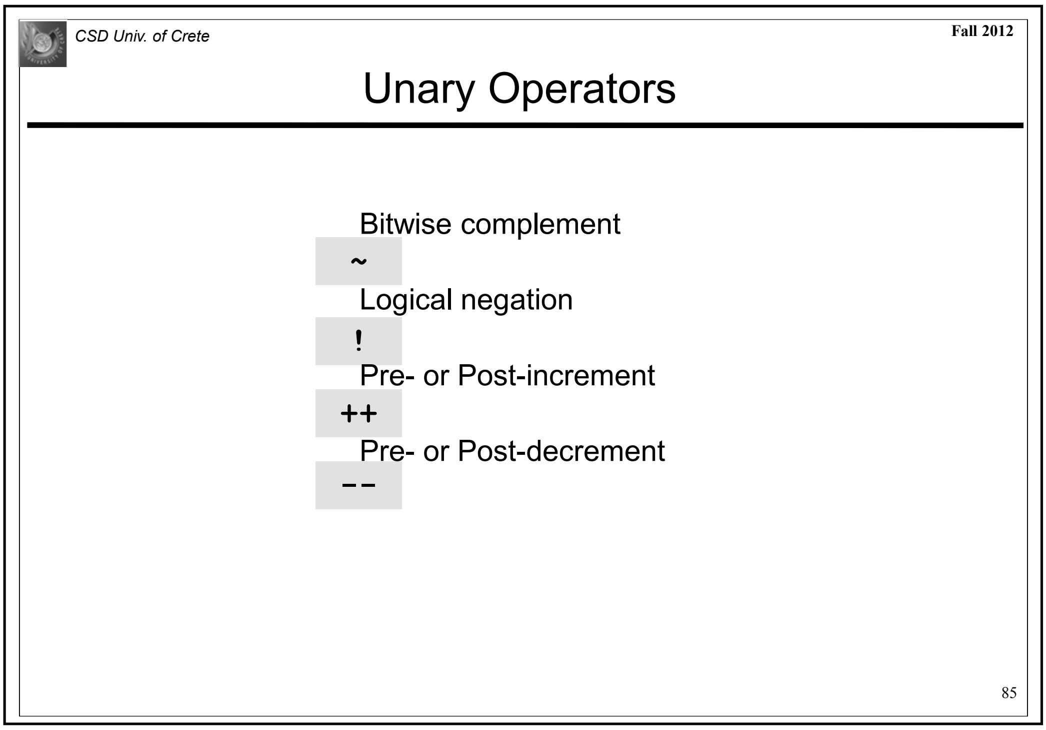
Some operators are “switch hitters”

Unary Operators

`()` Group expression

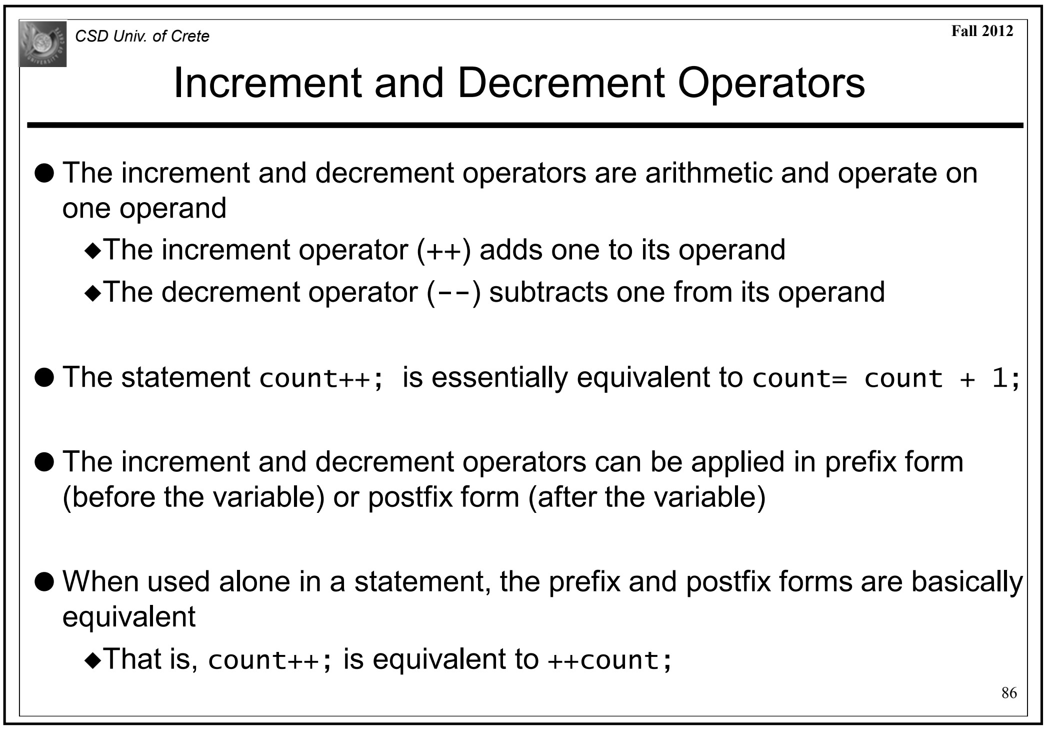
`+` Unary plus

`-` Unary minus



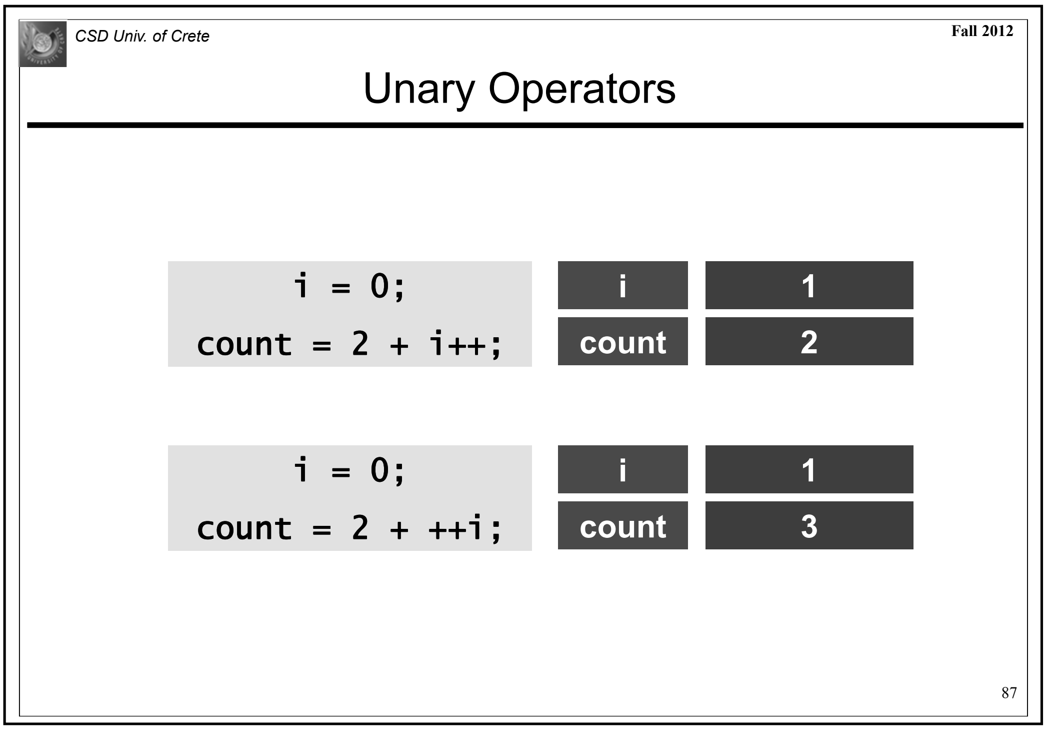
Unary Operators

~	Bitwise complement
!	Logical negation
++	Pre- or Post-increment
--	Pre- or Post-decrement



Increment and Decrement Operators

- The increment and decrement operators are arithmetic and operate on one operand
  - The increment operator (++) adds one to its operand
  - The decrement operator (`--`) subtracts one from its operand
- The statement `count++;` is essentially equivalent to `count= count + 1;`
- The increment and decrement operators can be applied in prefix form (before the variable) or postfix form (after the variable)
- When used alone in a statement, the prefix and postfix forms are basically equivalent
  - That is, `count++;` is equivalent to `++count;`



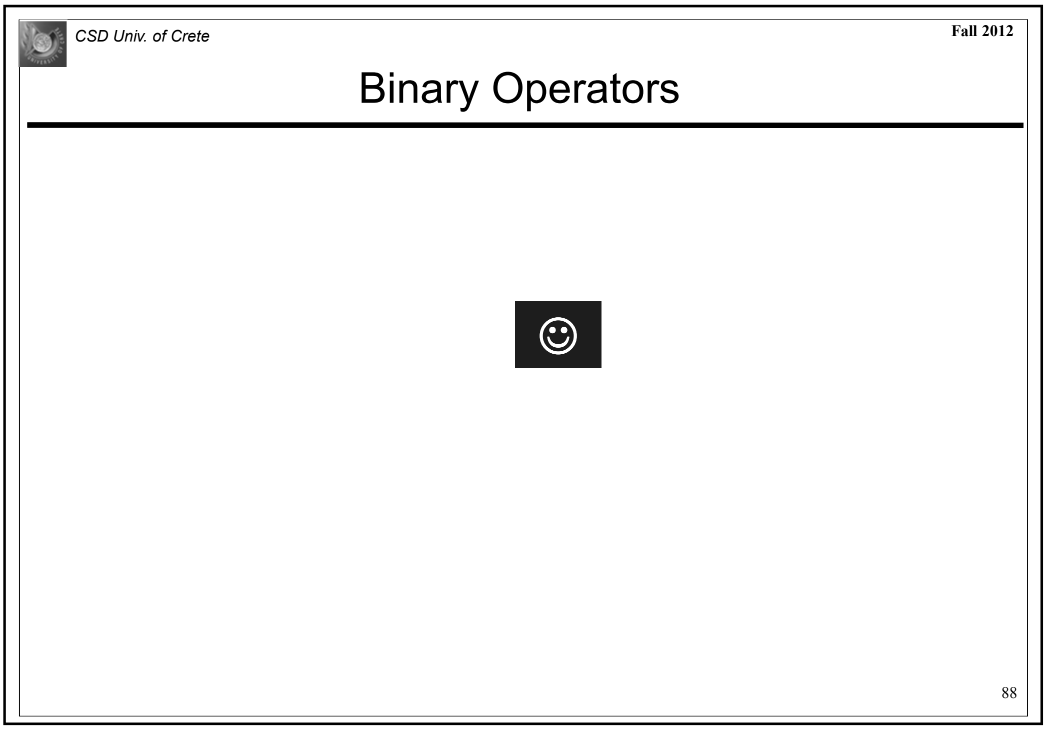
Unary Operators

```
i = 0;
count = 2 + i++;
```

i	1
count	2

```
i = 0;
count = 2 + ++i;
```

i	1
count	3

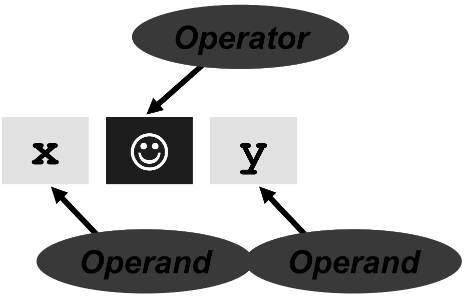


Binary Operators

☺

Binary Operators

Binary Operators



Binary Operators

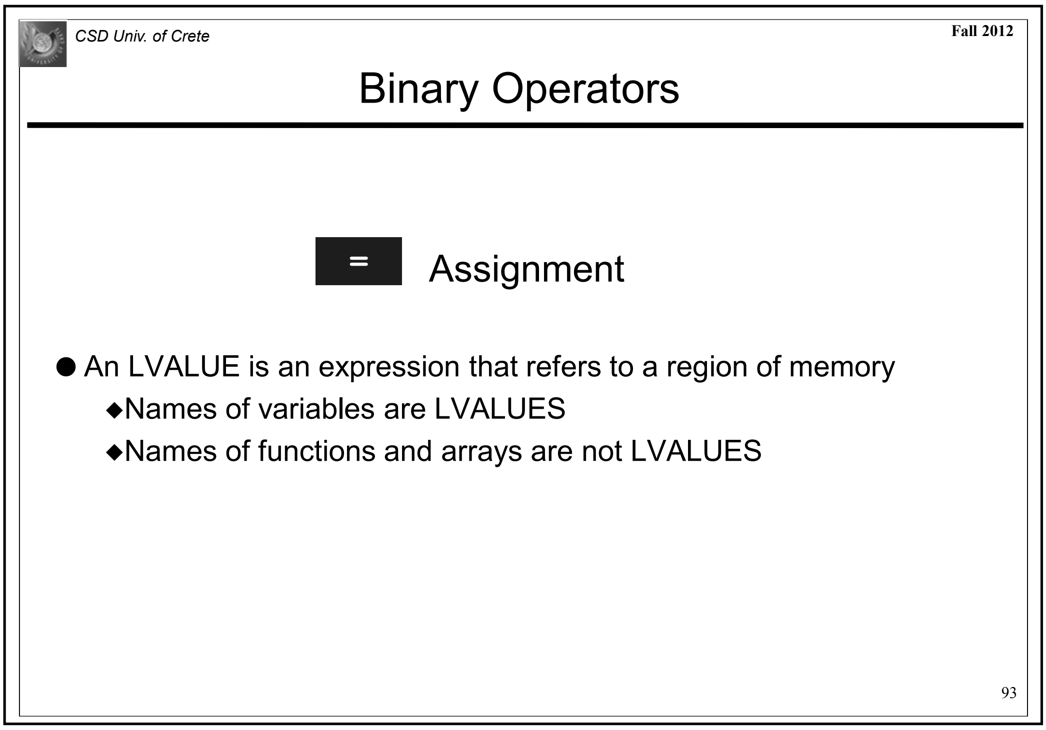
Additive & Multiplicative

+	Plus
−	Minus
*	Multiply
/	Divide
%	Remainder

Binary Operators

= Assignment

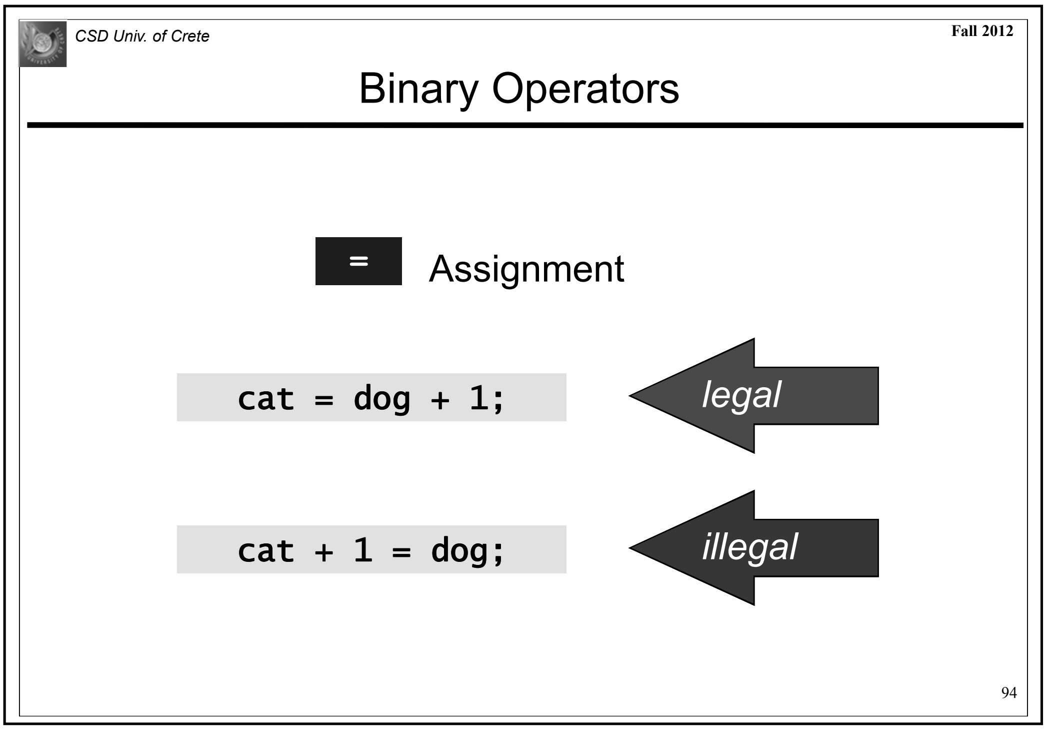
- Assignment is a binary operator in Java
- The left-hand operand of an assignment must be an LVALUE (left value)



Binary Operators

= Assignment

- An LVALUE is an expression that refers to a region of memory
  - ◆Names of variables are LVALUES
  - ◆Names of functions and arrays are not LVALUES



Binary Operators

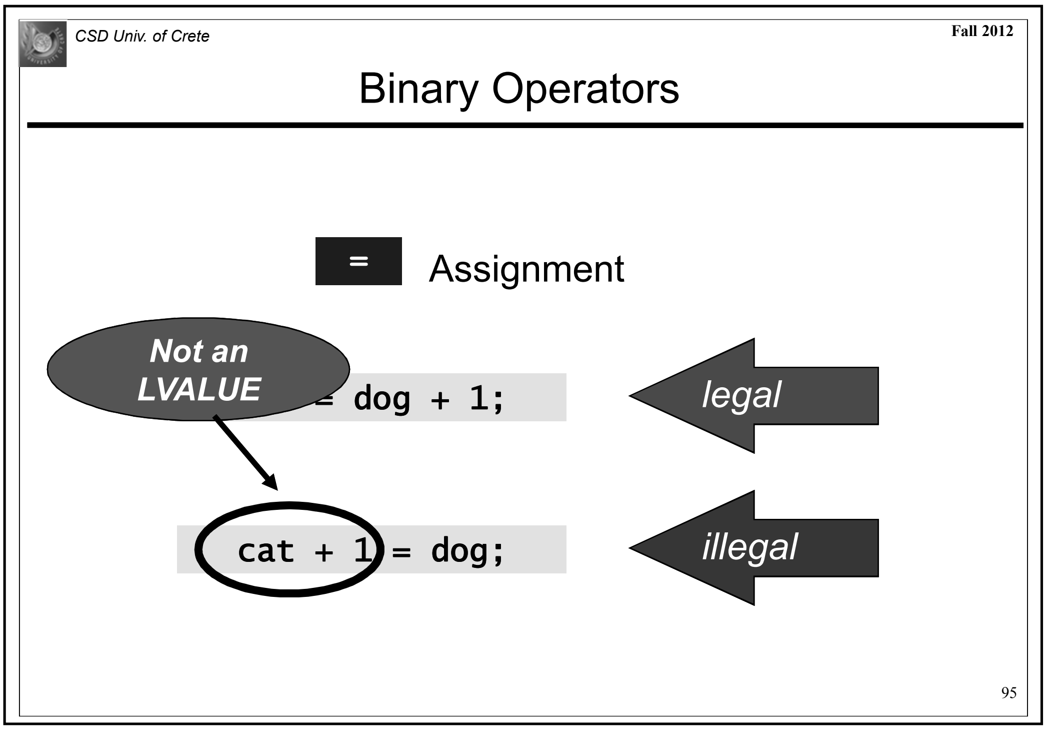
= Assignment

```
cat = dog + 1;
```

legal

```
cat + 1 = dog;
```

illegal



Binary Operators

= Assignment

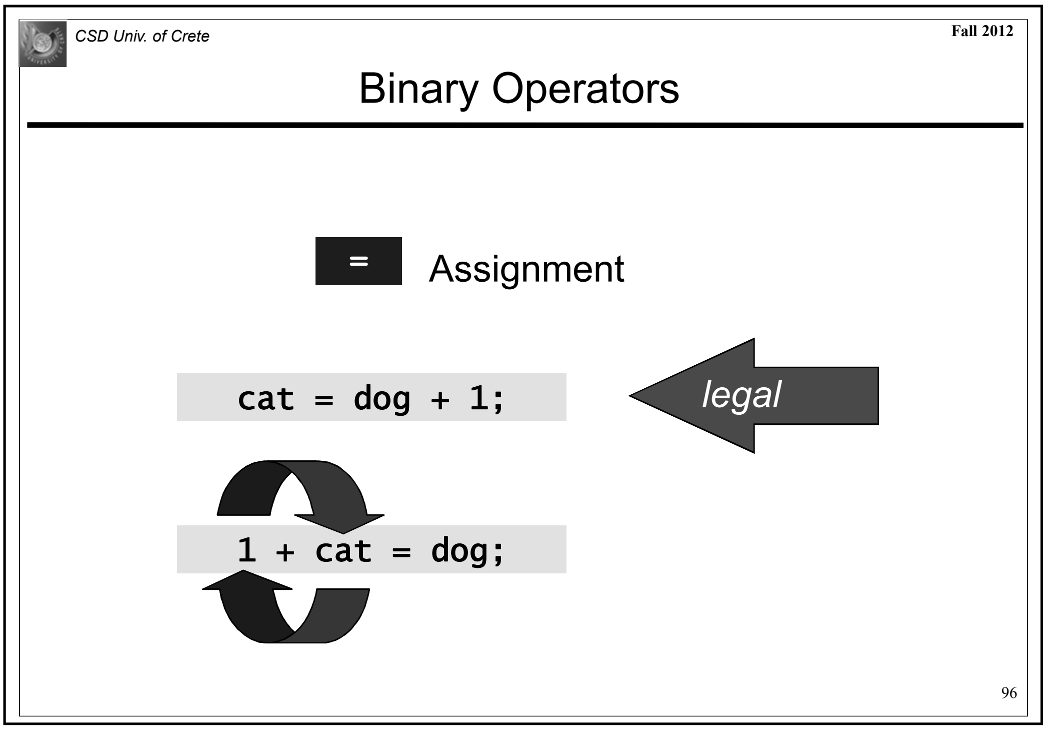
Not an LVALUE

```
= dog + 1;
```

legal

```
cat + 1 = dog;
```

illegal



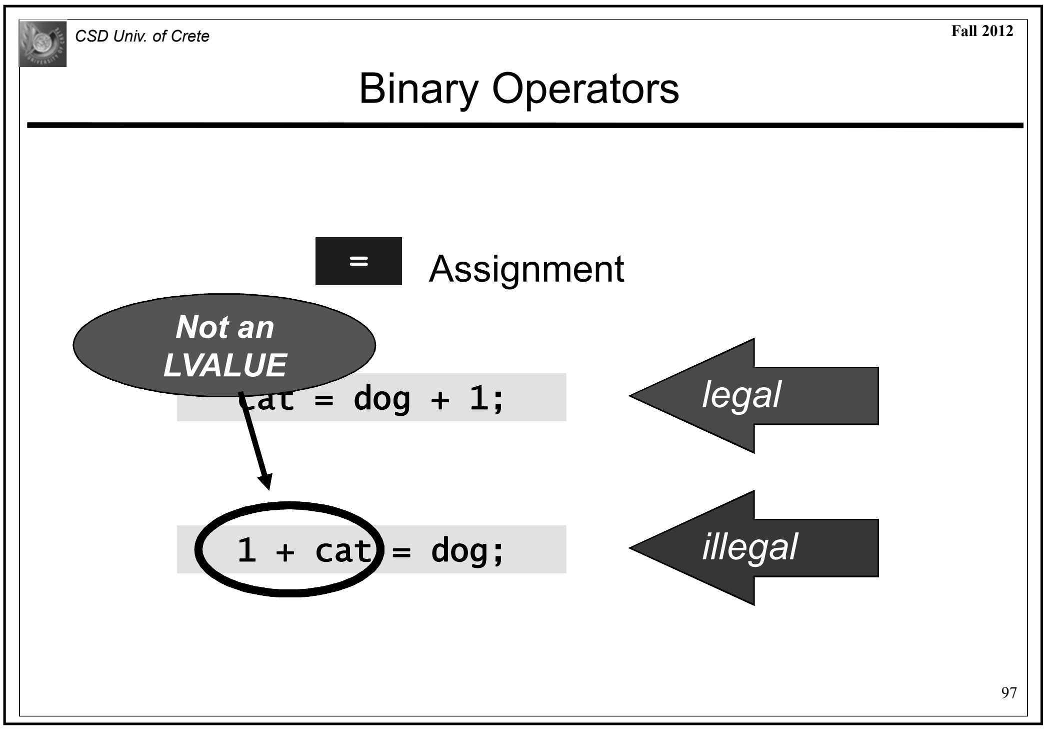
Binary Operators

= Assignment

```
cat = dog + 1;
```

legal

```
1 + cat = dog;
```



CSD Univ. of Crete

Fall 2012

Binary Operators

= Assignment

Not an LVALUE

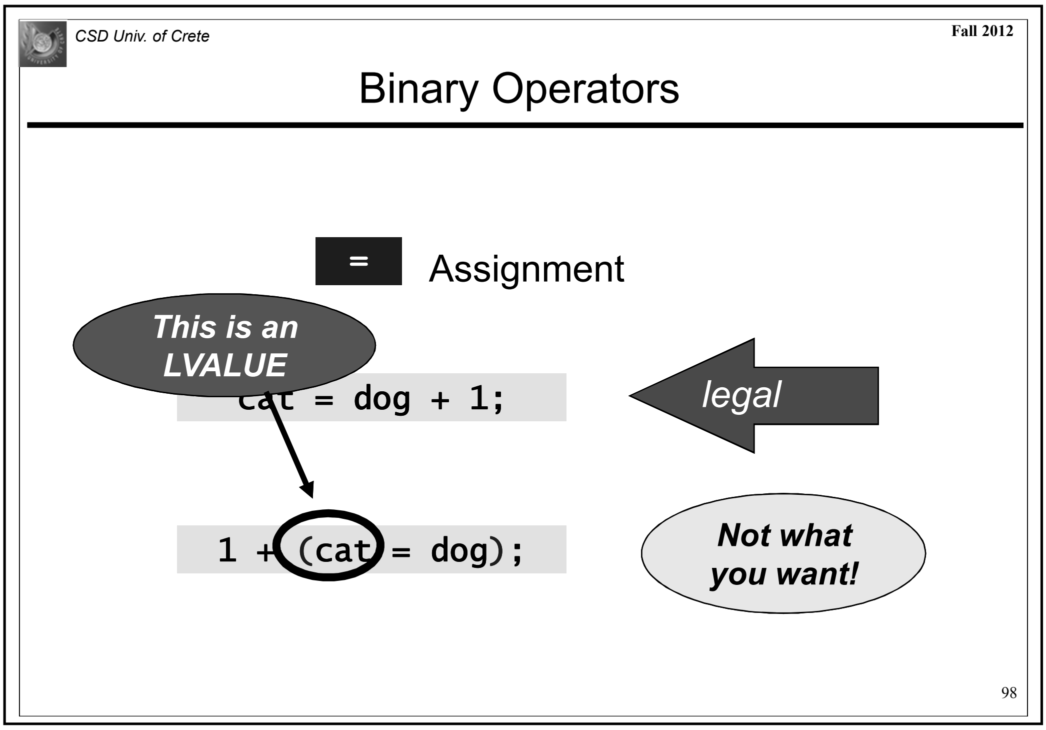
```
cat = dog + 1;
```

legal

```
1 + cat = dog;
```

illegal

97



CSD Univ. of Crete

Fall 2012

Binary Operators

= Assignment

This is an LVALUE

```
cat = dog + 1;
```

legal

```
1 + (cat = dog);
```

Not what you want!

98

Binary Operators

```
class Test {
    public static void main(String[] args) {
            int result, val_1, val_2;
            result = (val_1 = 1) + (val_2 = 2);
            System.out.println("val_1 = "+val_1);
            System.out.println("val_2 = "+val_2);
            System.out.println("result = "+result);
      }
}
```

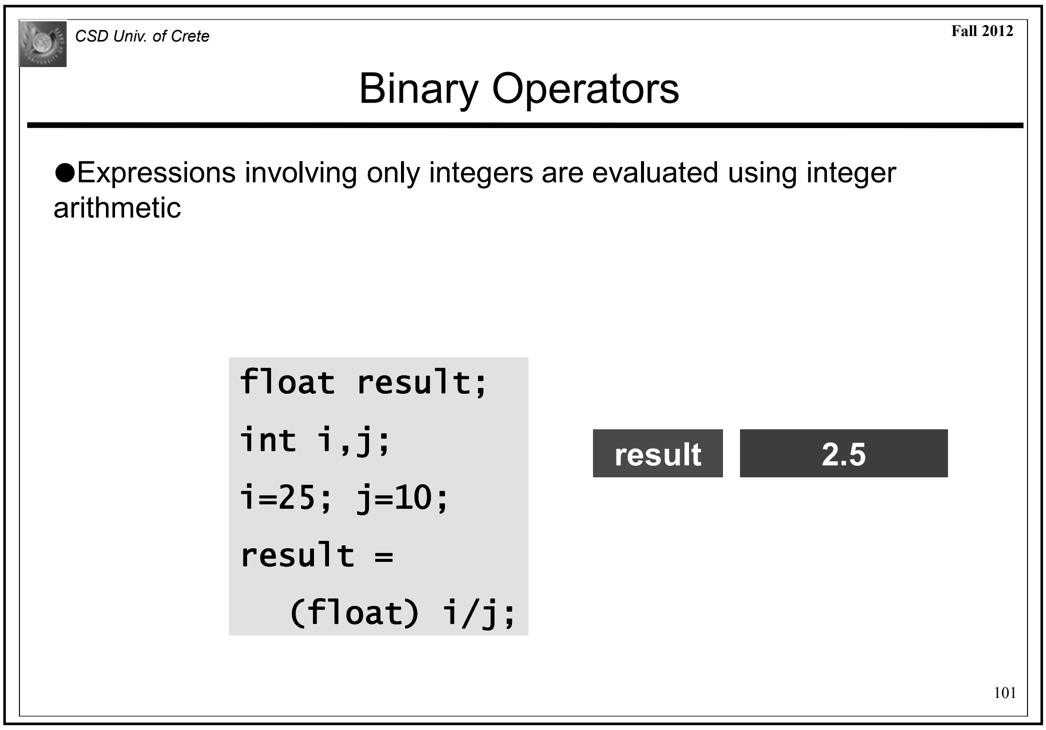
```
val_1 = 1
val_2 = 2
result = 3
```

Binary Operators

- Expressions involving only integers are evaluated using integer arithmetic

```
float result;
int i,j;
i=25; j=10;
result = i/j;
```

result	2.0

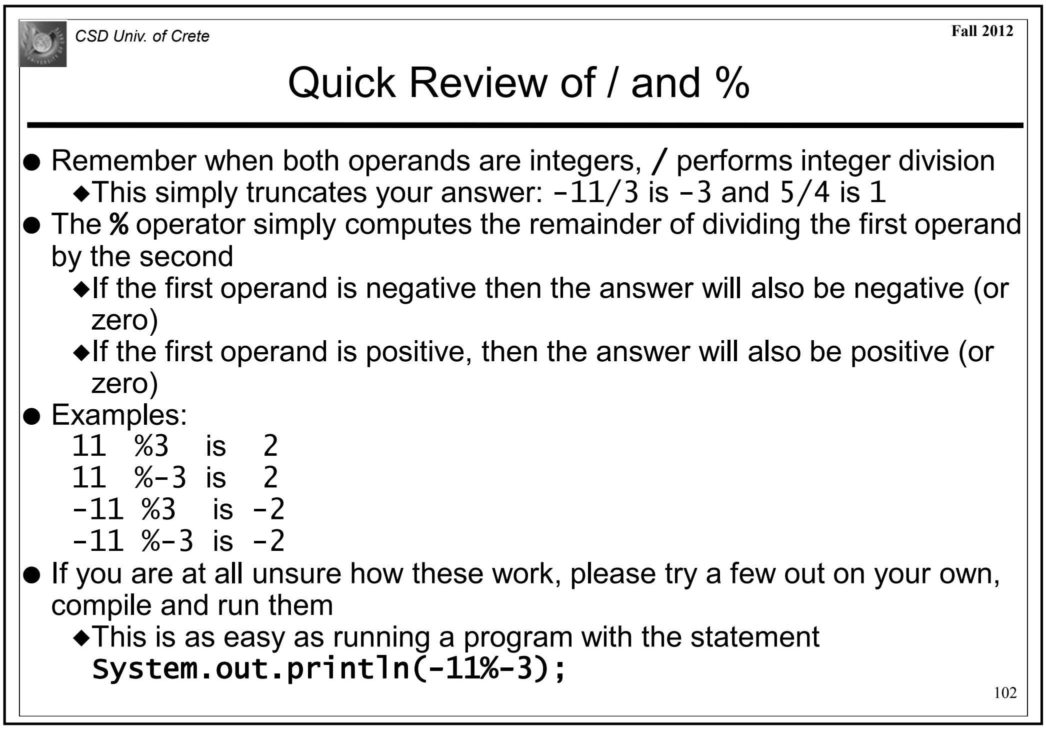


Binary Operators

- Expressions involving only integers are evaluated using integer arithmetic

```
float result;
int i,j;
i=25; j=10;
result =
   (float) i/j;
```

result	2.5



Quick Review of / and %

- Remember when both operands are integers, `/` performs integer division
  - ◆This simply truncates your answer: `-11/3` is `-3` and `5/4` is `1`
- The `%` operator simply computes the remainder of dividing the first operand by the second
  - ◆If the first operand is negative then the answer will also be negative (or zero)
  - ◆If the first operand is positive, then the answer will also be positive (or zero)
- Examples:

```
11  %3   is  2
11  %-3  is  2
-11 %3   is -2
-11 %-3  is -2
```

- If you are at all unsure how these work, please try a few out on your own, compile and run them
  - ◆This is as easy as running a program with the statement `System.out.println(-11%-3);`

Binary Operators

Assignment

Assign sum

`+=`

Assign difference

`-=`

Assign product

`*=`

Assign quotient

`/=`

Assign remainder

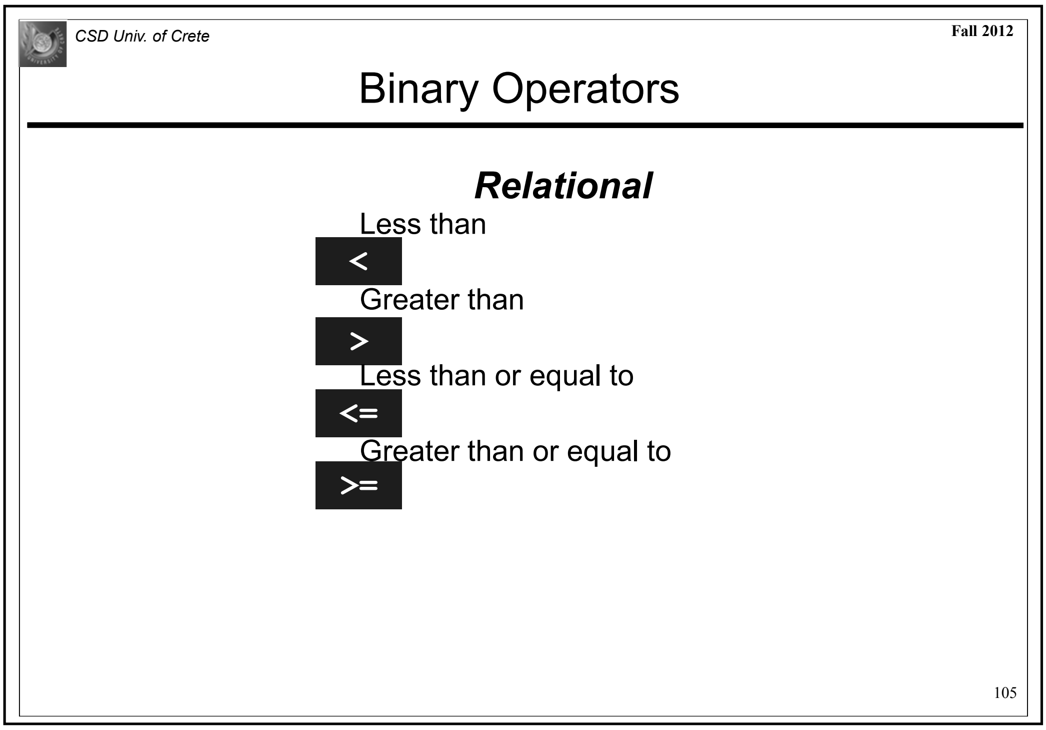
`%=`

Binary Operators

Assignment

Compound operators provide a convenient shorthand

```
int i;
i = i + 5;
i += 5;
```



Binary Operators

Relational

Less than

<

Greater than

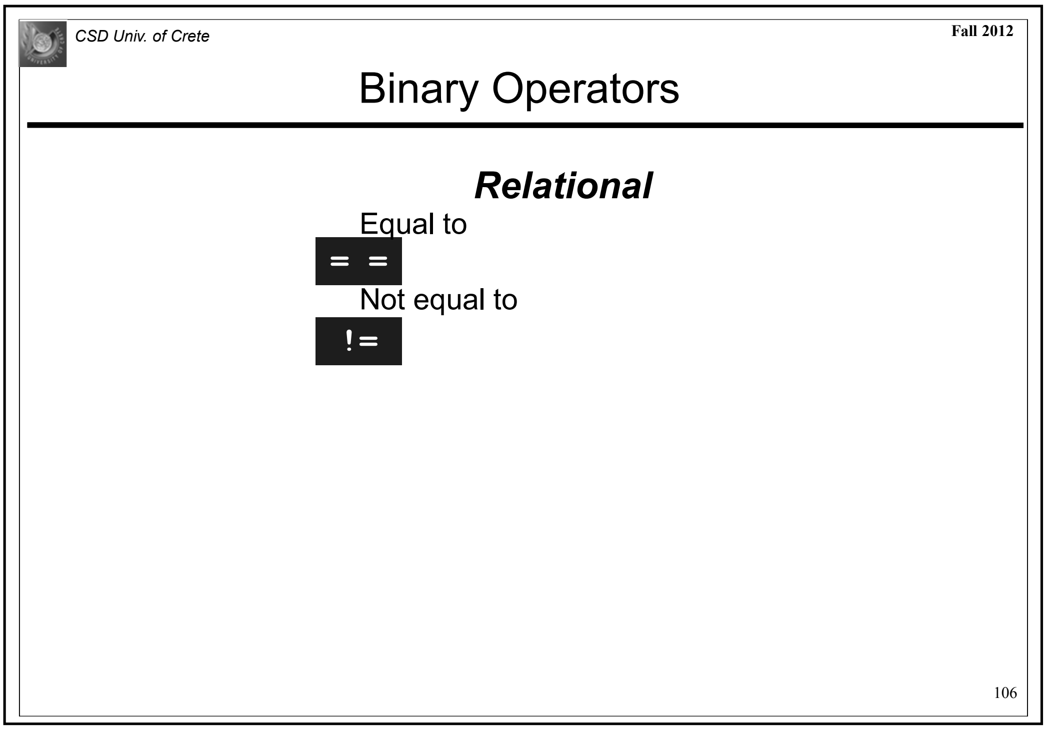
>

Less than or equal to

<=

Greater than or equal to

>=



Binary Operators

Relational

Equal to

= =

Not equal to

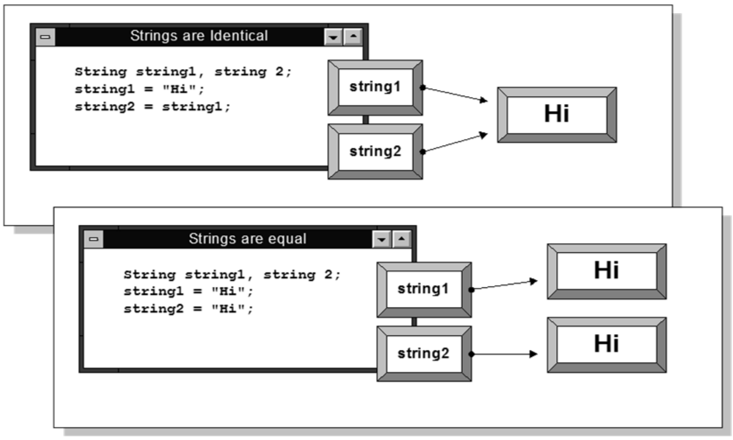
!=

Equality Testing

- Identity or simple equality
  - ◆ Two variables refer to the same object or String
- Value equality
  - ◆ Two object variables have the same contents
- Test for identity using the equality operators
- For value equality use the `equals()` method defined on Java **Object**s
- Example:

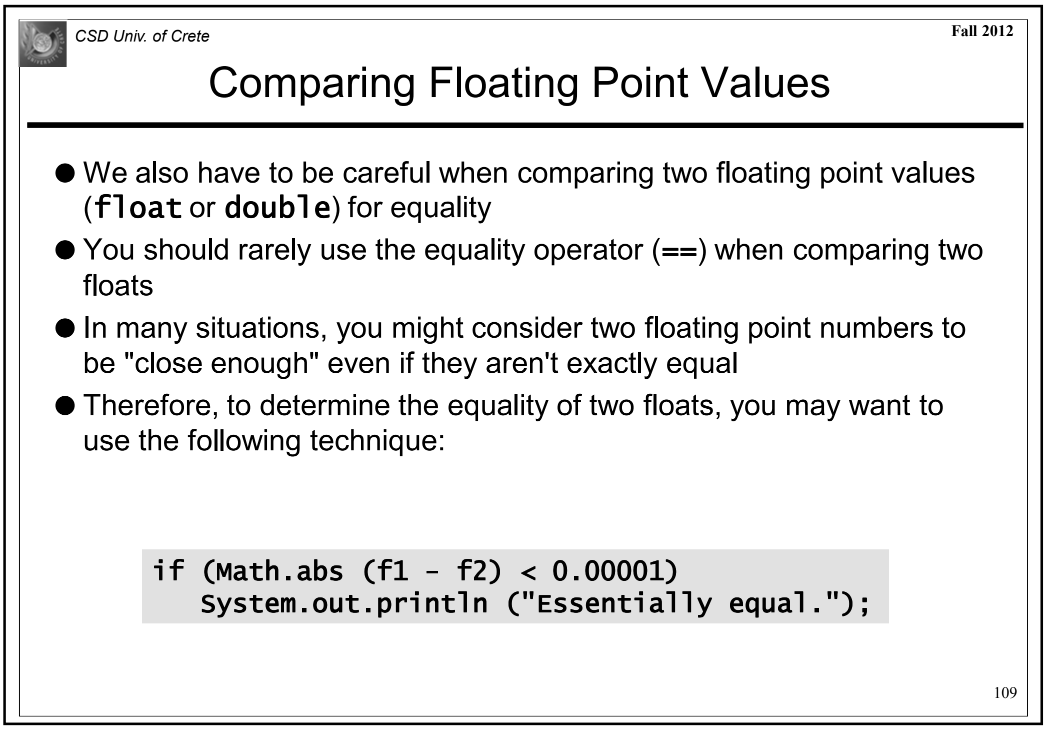
```
if ( nameString.equals(“Hi”) ) { }
```

Equality vs. Identity



```
String string1, string 2;
string1 = "Hi";
string2 = string1;
```

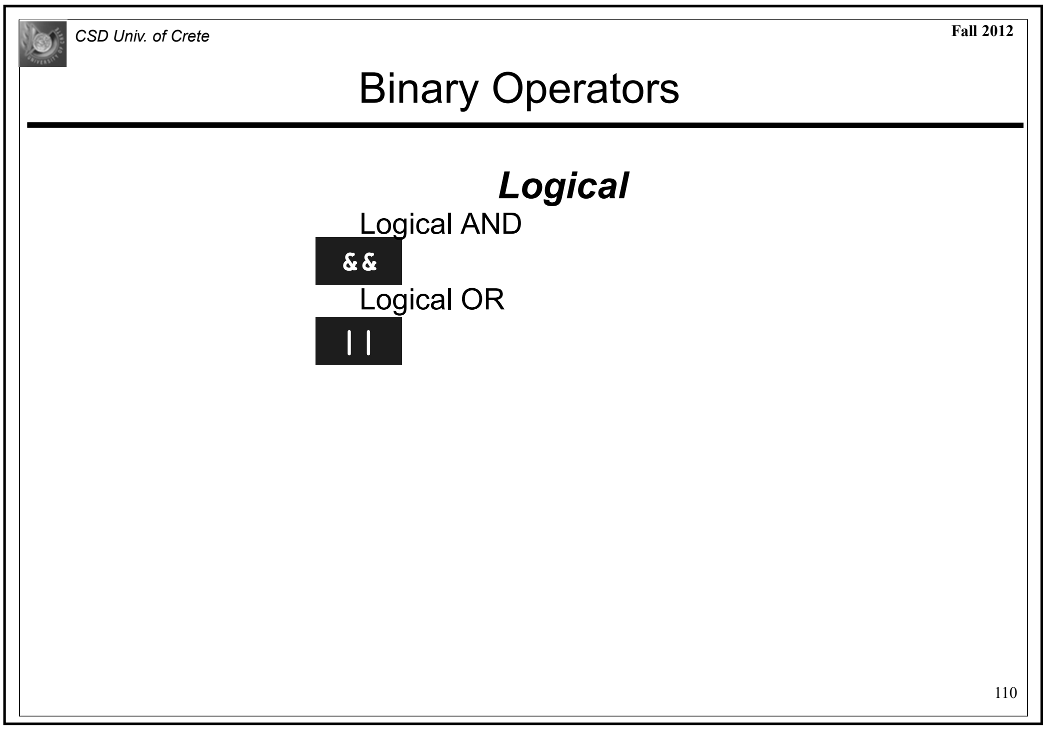
```
String string1, string 2;
string1 = "Hi";
string2 = "Hi";
```



Comparing Floating Point Values

- We also have to be careful when comparing two floating point values (`float` or `double`) for equality
- You should rarely use the equality operator (==) when comparing two floats
- In many situations, you might consider two floating point numbers to be "close enough" even if they aren't exactly equal
- Therefore, to determine the equality of two floats, you may want to use the following technique:

```
if (Math.abs (f1 - f2) < 0.00001)
   System.out.println ("Essentially equal.");
```



Binary Operators

Logical

Logical AND

`&&`

Logical OR

`||`

Binary Operators

- Expressions connected by && and || are evaluated from left to right

This never gets evaluated!

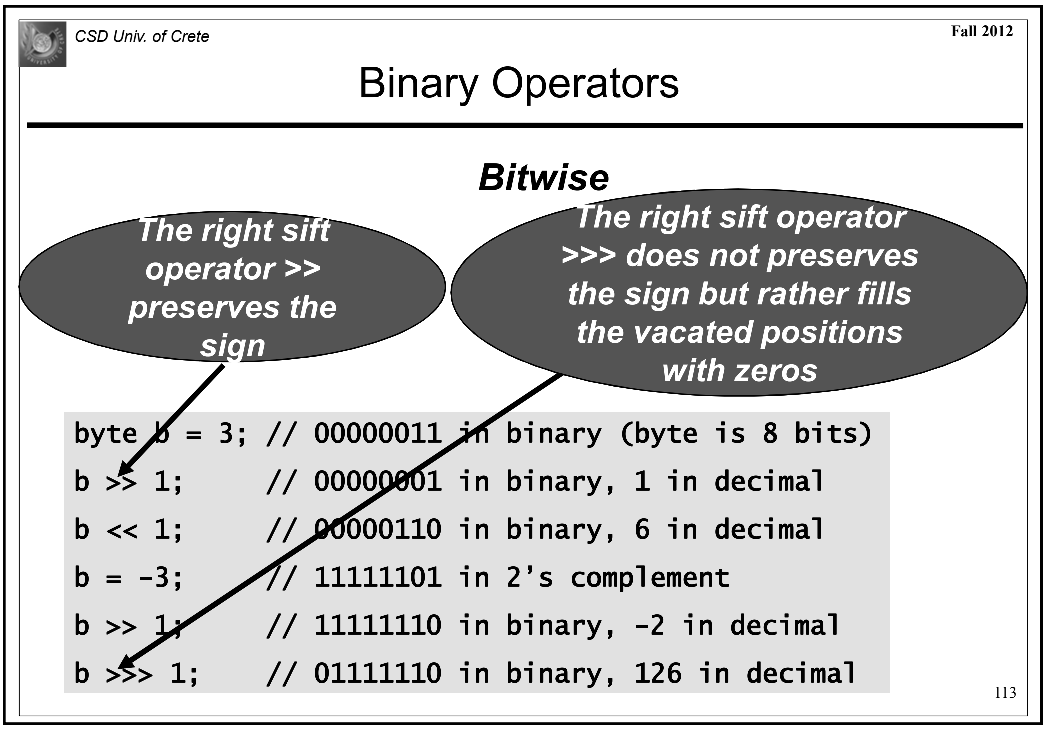
```
class Test {
  public static void main(String[] args) {
    int i=0;
    System.out.println("Test:" + ((2<3) || (0<i++)));
    System.out.println("I:" + i);
  }
}
```

```
Test:true
I:0
```

Binary Operators

Bitwise

<<	Shift left
>>	Shift right
>>>	Shift right with zero extension
~	One’s complement
&	Bitwise AND
^	Bitwise XOR
|	Bitwise OR



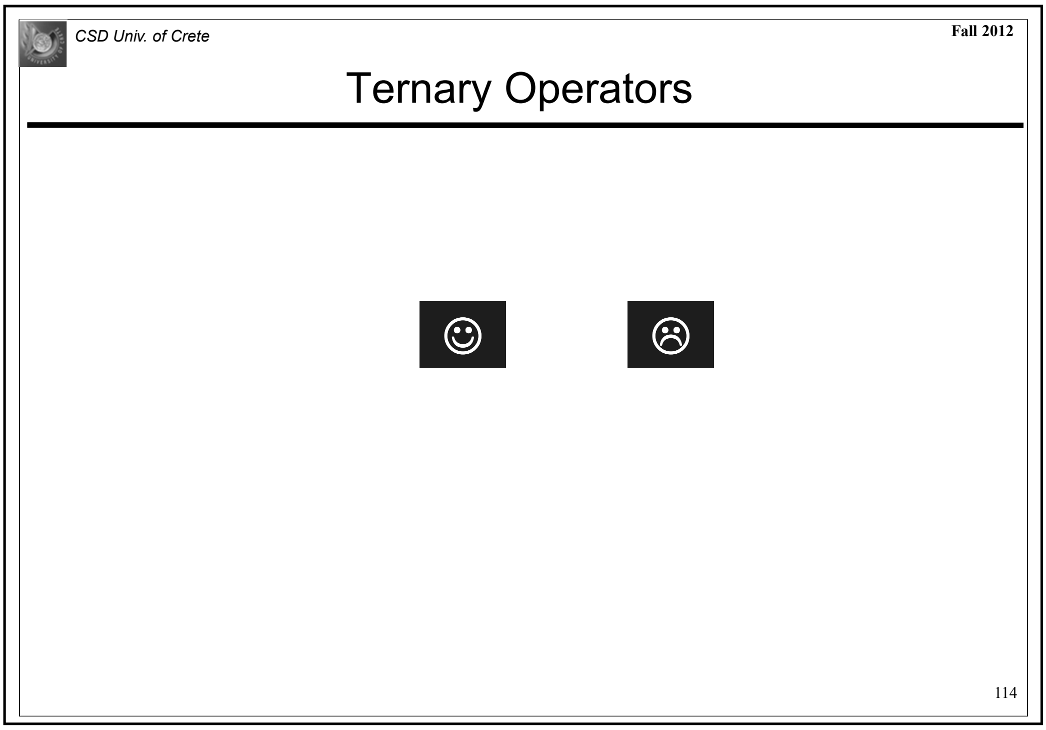
Binary Operators

Bitwise

The right sift operator >> preserves the sign

The right sift operator >>> does not preserves the sign but rather fills the vacated positions with zeros

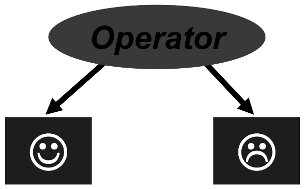
```
byte b = 3; // 00000011 in binary (byte is 8 bits)
b >> 1;     // 00000001 in binary, 1 in decimal
b << 1;     // 00000110 in binary, 6 in decimal
b = -3;     // 11111101 in 2’s complement
b >> 1;     // 11111110 in binary, -2 in decimal
b >>> 1;    // 01111110 in binary, 126 in decimal
```



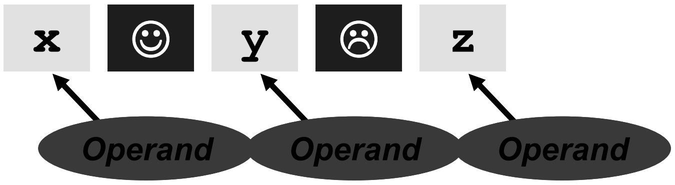
Ternary Operators

☺ ☹

Ternary Operators



Ternary Operators



Ternary Operators

Conditional

"if *a* then *x*, else *y*"

```
a?x:y
```

```
result = (x<y) ? x : y;
```

Mixing Operators

```
class Test {
  public static void main(String[] args) {
    char cv;
    int iv1 = 64;
    cv = (char) iv1;
    System.out.println("cv:" + cv);
    System.out.println("iv1:" + iv1);
  }
}
```

@

64

```
cv:@
iv1:64
```

Mixing Operators

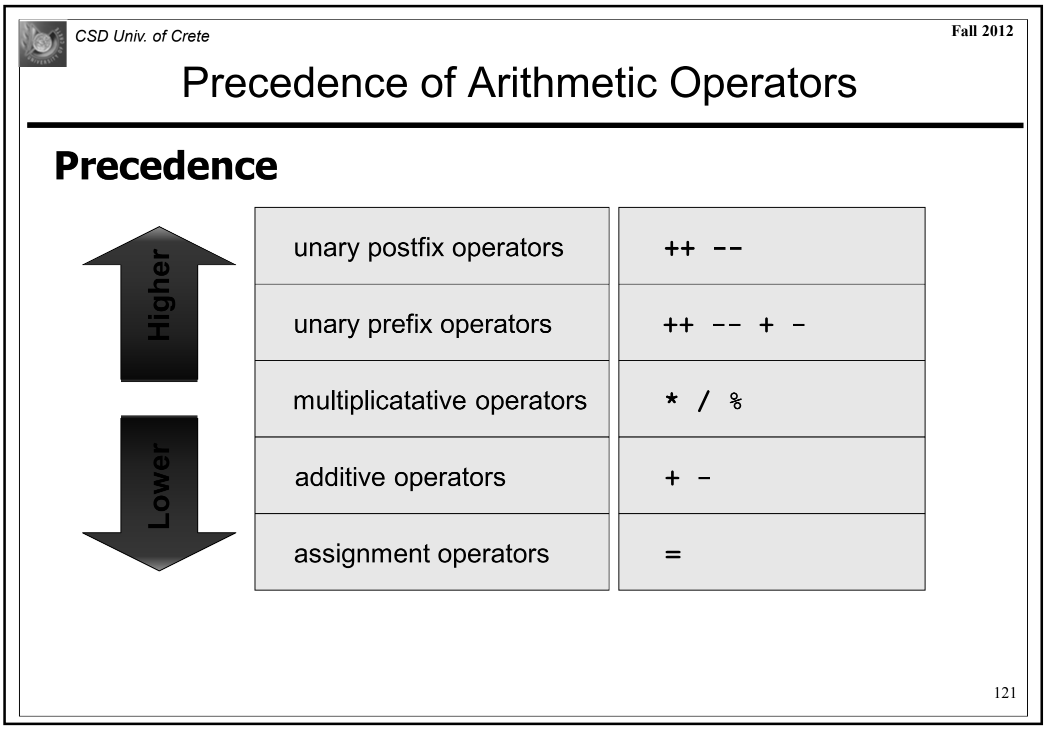
```
class Test {
  public static void main(String[] args) {
    double fv1, fv2;
    int iv1 = 123;
    fv1 = iv1/50;
    fv2 = iv1/50.0;
    System.out.println("fv1:" + fv1);
    System.out.println("fv2:" + fv2);
  }
}
```

Floating point constants are assumed to be double, by default!

```
fv1:2.0
fv2:2.46
```

Operator Precedence

- The order in which operands are evaluated in an expression is determined by a well-defined precedence hierarchy
  - ◆Precedence is the same as order of evaluation; Every operand will be evaluated before operation
- Operators at the same level of precedence are evaluated according to their associativity. Java guarantees left-to-right evaluation (unlike C) ;
  - ◆All binary operators except assignment are left-associative
  - ◆Assignment is right-associative
- Parentheses can be used to force precedence

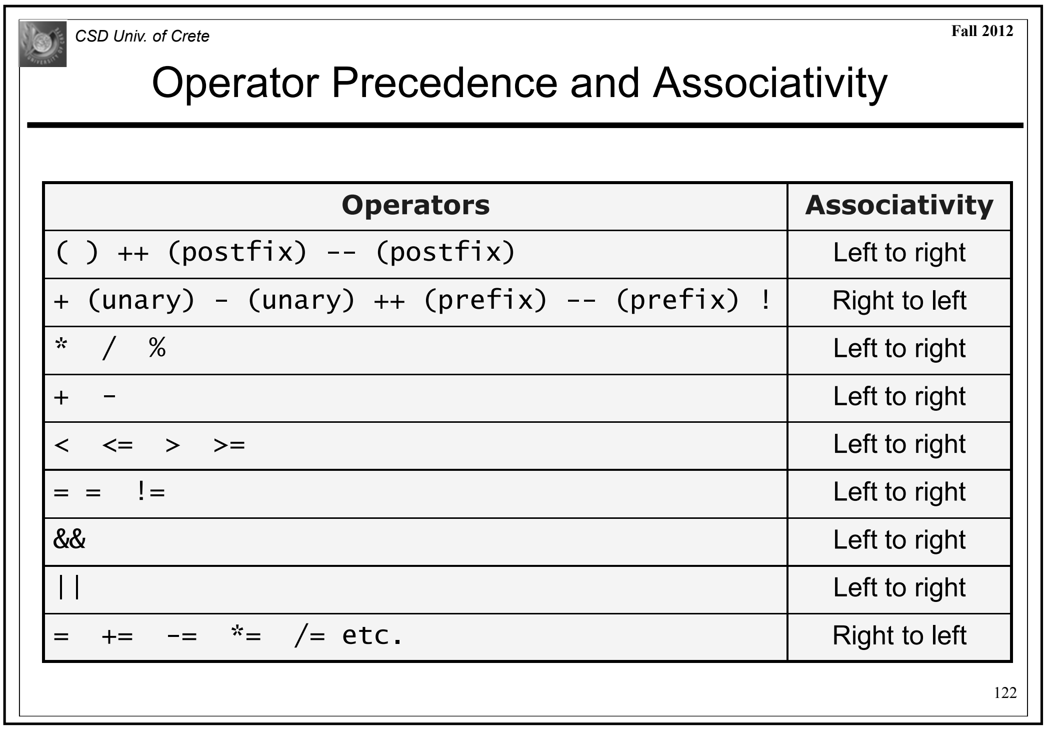


Precedence of Arithmetic Operators

Precedence

Higher
Lower

unary postfix operators	`++ --`
unary prefix operators	`++ -- + -`
multiplicatative operators	`* / %`
additive operators	`+ -`
assignment operators	`=`



Operator Precedence and Associativity

Operators	Associativity
`( ) ++ (postfix) -- (postfix)`	Left to right
`+ (unary) - (unary) ++ (prefix) -- (prefix) !`	Right to left
`* / %`	Left to right
`+ -`	Left to right
`< <= > >=`	Left to right
`= =  !=`	Left to right
&&	Left to right
`||`	Left to right
`= += -= *= /= etc.`	Right to left

Operator Precedence

Expression	Result
`2 + 3 * 4 / 2`	8
`3 * 13 + 2`	41
`(3 * 13) + 2`	41
`3 * (13 + 2)`	45
`4 * (11 - 6) * (-8 + 10)`	40
`(5 * (4 - 1)) / 2`	7
`5 + 12 / 5 - 10 % 3`	6

3 1 4 2

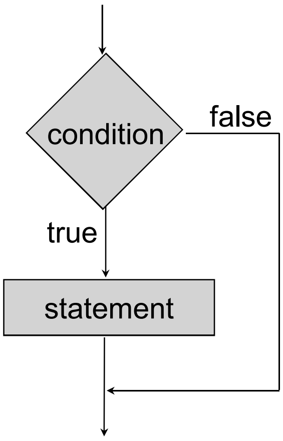
Java Control Structures

The IF Statement

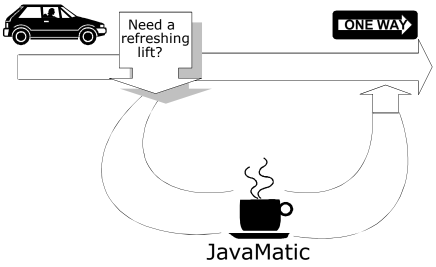
- The Java **if** statement has the following syntax:

```
if (condition) {
    statement(s);
  }
```

  - ◆If the boolean condition is true, the statement is executed; if it is false, the statement is skipped
- Selection statements provides basic decision making capabilities



The IF Illustrated



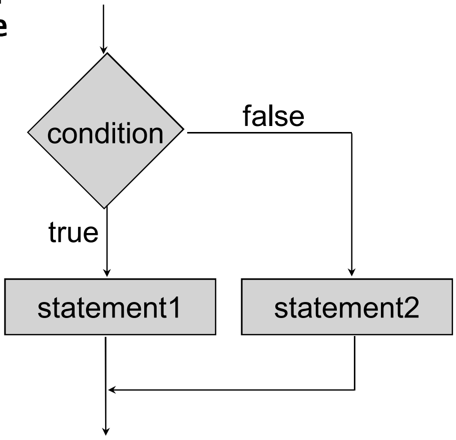
```
The if statement

if ( youNeedALift( ) )
{
      tryJavaMatic(chickenSoup);
}
```

The IF-ELSE Statement

- An **else** clause can be added to an **if** statement to make it an **if-else** statement:

```
    if (condition) {
        statement(s)1;}
    else {
        statement(s)2;}
```

  - ◆If the condition is true, statement1 is executed; if the condition is false, statement2 is executed
- The body of an if statement or else clause can be another if statement (nested **if** statement)
- Note: an **else** clause is matched to the last unmatched **if** (no matter what the indentation implies)



A Warning…

WRONG!

```
if( i == j )
    if ( j == k )
      System.out.print(
         “i equals k”);
     else
     System.out.print(
      “i is not equal  to
    j”);
```

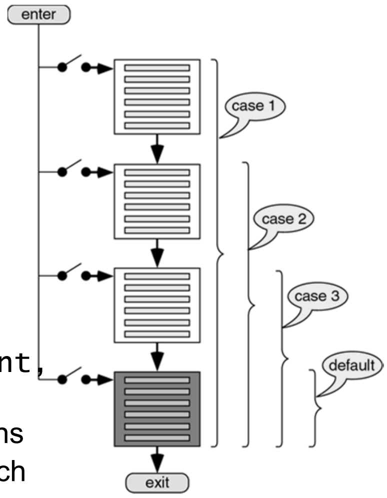
CORRECT!

```
if( i == j ) {
 if ( j == k )
 System.out.print(
     “i equals k”);
}
else
 System.out.print(“i
 is not equal to j”);
   // Correct!
```

The SWITCH Statement

- Multiple selections with **switch** statement*:*

```
switch( expression ) {
   case value1 :
            statement(s); break;
   case value2 :
            statement(s); break;
     ...
   default :
            statement(s); break;
}
```



- Expression must be integral type: `short`,`int`, `byte` or `char` but can not be a `String`
  - ◆the cases are actual values, not conditions
- **break** passes the control to the end of switch
- **default** catches all other cases

Points to Note on Switch

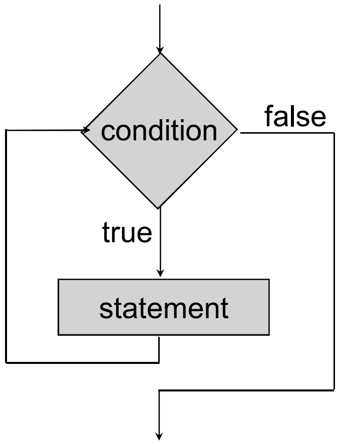
- The name of the control variable is placed in parentheses
- Following the word **switch** and control variable, you have all the cases set up in a single block (i.e., set of curly braces)
- All the reserved words (**switch**, **case**, **default**) start with a lower case letter
- Each case ends with a colon character "**:**" (note: not a semi-colon)
- Each statement can be a simple statement or a compound statement (i.e., block)
- Normally each case (including the last one) ends with a **break** statement
- Each case label has to be unique
- There can be only one "default" label

The WHILE Statement

- A **while** statement has the following syntax:

```
Initialize the conditions
while (condition)
        statement(s);
```

  - ◆If the condition is true, the statement is executed; then the condition is evaluated again
  - ◆Statements perform the loop and change the conditions
- The statement is executed over and over until the condition becomes false (repetition)



Points to Note on WHILE

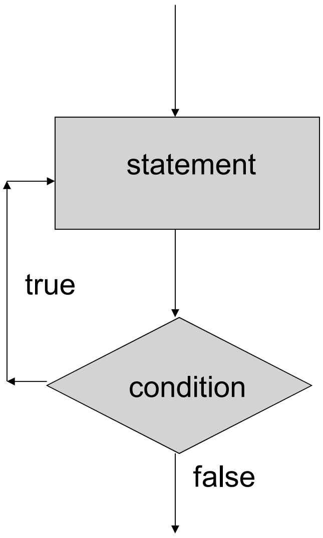
- If the condition of a **while** statement is false initially, the statement is never executed
  - ◆Therefore, we say that a **while** statement executes zero or more times
- The body of a **while** loop must eventually make the condition false
  - ◆If not, it is an infinite loop, which will execute until the user interrupts the program
- This is a common type of logical error
  - ◆always double check that your loops will terminate normally

The DO-WHILE Statement

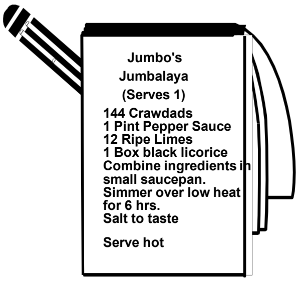
- A **do while** statement has the following syntax:

```
Initialize the conditions
do {
    statement(s);
   } while (conditions);
```

  - ◆The statement is executed; then if the condition is true, the statement is executed again
  - ◆Statements perform the loop and change the conditions

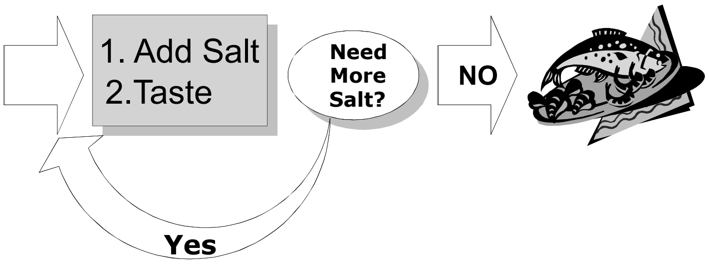


The DO-WHILE Illustrated



Jumbo's
Jumbalaya
(Serves 1)
144 Crawdads
1 Pint Pepper Sauce
12 Ripe Limes
1 Box black licorice
Combine ingredients in small saucepan.
Simmer over low heat for 6 hrs.
Salt to taste
Serve hot

```
do
{
   AddSalt( );
   Taste( );
} while ( needsMoreSalt( ));
```

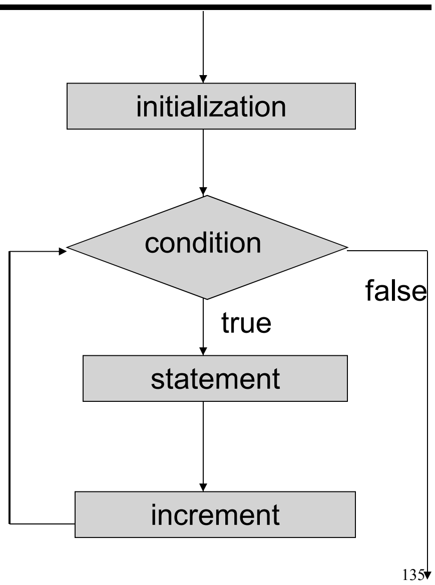


The FOR Statement

- A **for** statement has the following syntax:

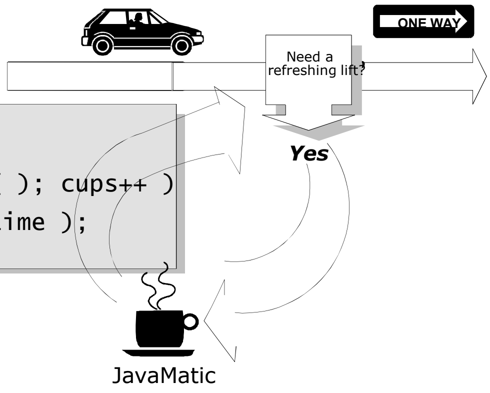
```
for (initialization;
     condition;
     increment)
     {statement(s);}
```

- Loop condition: 3 parts, separate with ;
  - ◆The initialization section: always executes, only once
  - ◆The boolean test: evaluated before each iteration
  - ◆The update expression: evaluated after loop body

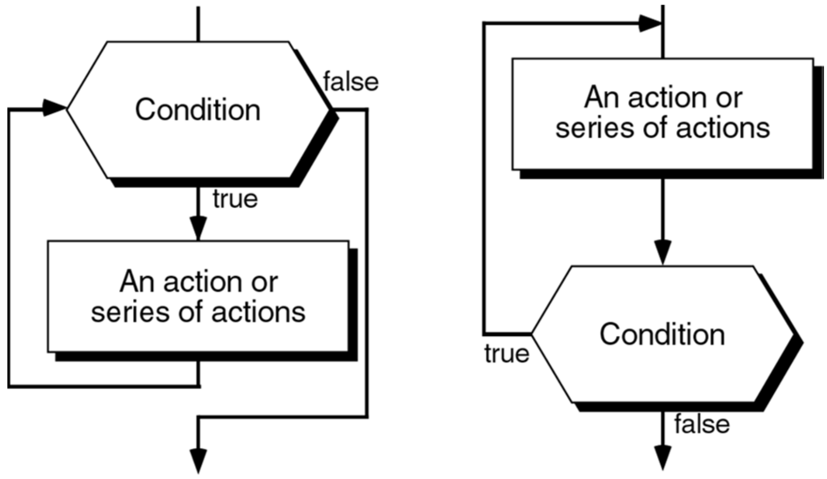


The FOR Illustrated

```
The for loop

for ( cups = 0; youNeedALift( ); cups++ )
{
     tryJavaMatic( licoriceLime );
}
```



Looping : Various Schemes


Condition
false
true
An action or series of actions
An action or series of actions
true
Condition
false

(a) Pretest Loop (b) Post-test Loop

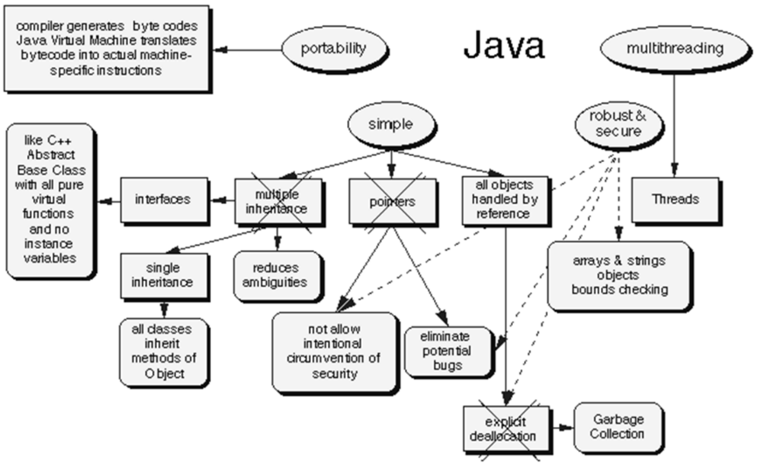
Java vs. C/C++ Stand-alone Programs

- The **`main`** is required in both languages for a stand-along program
- C++ ➔ `int main (int argc, char* argv)`
  - has understood arguments **`argc`** and **`argv`**
    - **`argc`** = number of command line arguments in invocation
    - **`argv`** = pointer to string array of arguments **`argv[0]=prg`** name
  - **`main`** may return a value using **`return( )`** or **`exit( )`**
  - by default is **`int`**
- Java ➔ `public static void main(String argv[ ])`
  - has one single argument, an array of strings, conventionally named `args` or `argv`
    - In Java, the array itself is an object and it is referenced through the array name
    - The length of a Java array is in `arrayName.length`
    - For example `argv.length` if `argv` is parameter name
  - main must be declared **`void`**
  - It can't return a value, instead of **`return()`** use **`System.exit();`**

Differences between Java and C/C++

- Java
  - ◆Single inheritance
  - ◆C data type not supported
    - struct, union, pointer
  - ◆Command line arguments
    - args
  - ◆String
    - First-class object
  - ◆Exception handling
    - Try-Catch-Finally
  - ◆Garbage collection
  - ◆No operator overloading
- C++
  - ◆Multiple inheritance
  - ◆C data type supported
  - ◆Command line arguments
    - argc, argv
  - ◆String
    - character array
  - ◆Exception handling
    - Try-Catch
  - ◆No garbage collection
  - ◆Operator overloading

Java Overview


compiler generates byte codes Java Virtual Machine translates bytecode into actual machine-specific instructions
portability
Java
multithreading
simple
robust & secure
like C++ Abstract Base Class with all pure virtual functions and no instance variables
interfaces
multiple inheritance
pointers
all objects handled by reference
Threads
single inheritance
reduces ambiguities
arrays & strings objects bounds checking
all classes inherit methods of Object
not allow intentional circumvention of security
eliminate potential bugs
explicit deallocation
Garbage Collection