

Some History

- 1993 Oak project at Sun
- 1995 Oak becomes Java; every major web player announces support
- 1996 Java 1.0 available
- 1997 (March) Java 1.1 some language changes and much larger library, including new event handling GUI model (AWT)
- 1997 (September) Java 1.2 beta including Swing GUI package
- 1998 (October) Java 1.2 final
- 2000 (April) Java 1.3 final

4/23/00

}

4/23/00

}

Design Goals Support secure, high-performance, robust applications running as-is on multiple platforms and over networks "Architecture-neutral", portable, allow dynamic updates and adapt to new environments Look enough like C++ for programmer comfort • Support object-oriented programming Support concurrency (multithreading) • •

Simplicity

4/23/00

4/23/00

Hello World in Java public class HelloWorld { public static void main (String [] args) { System.out.println("Hello World");

Classes · Everything in Java is a member of some class No external (global) functions or variables

- · Classes may contain methods and data members • Class members may be - non-static: one copy for each instance of the class
 - static: single copy associated with the class, not with any specific instances.

"Java has no functions. Object-oriented programming supersedes functional and procedural styles. Mixing the two styles just leads to confusion and dilutes the purity of an object-oriented language."

Gosling & McGilton Java White Paper

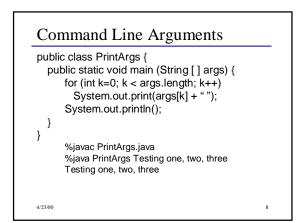
Hello World Revisited

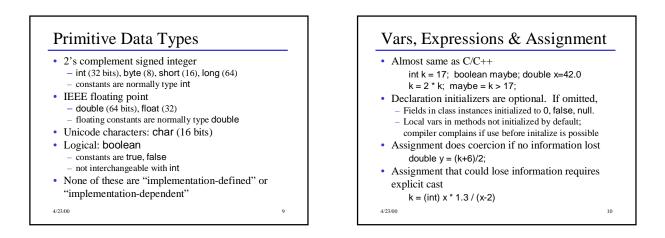
public class HelloWorld {
 public static void main (String [] args) {
 System.out.println("Hello World");

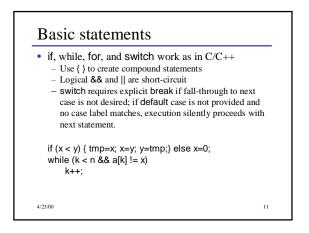
- }
- Every class may have a main method
- Execution begins in main of a designated class
- Class Xyzzy should be in file Xyzzy.java

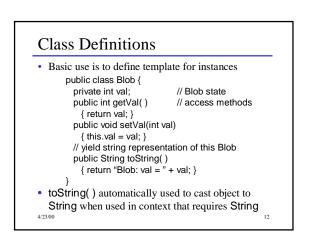
%javac HelloWorld.java %java Helloworld Hello World

4/23/00









Visibility

- Class members can be preceded by a qualifier to indicate accessibility
 - public accessible anywhere the class can be accessed
 private only accessible inside the class
 - If nothing is specified, the field can be referenced anywhere in the same package (more later).
 - protected same as package visibility, and also visible in classes that extend this class.

4/23/00

Instance Creation and References All variables that do not have a primitive type are references. Objects are only created by explicit allocation. Blob bob; // no blob allocated yet bob = new Blob(); // Blob allocated here bob.setVal(42); int k = bob.getVal(); System.out.println("bob is " + bob);

14

16

References and Methods

- Dot notation is used to select methods and fields; implicit dereference (no -> as in C/C++).
- No pointer arithmetic; no & operator to generate the address of arbitrary variable; can't create pointers from random bits.
 - "Java has no pointers"
- All method parameters are call-by-value (copy of primitive value or object reference)
- Methods can be overloaded (different methods with same name but different number or types of parameters).

4/23/00

Object Allocation

- A variable declared as class X has type "reference to X". No object is created by such a declaration.
- Declaration and object creation can be combined.
 Blob bob = new Blob();
- The constant null belongs to all reference types and refers to nothing.
- If reference r is null, then selecting a field from r (r.fieldname) throws a NullPointerException.
- Storage occupied by an object is dynamically reclaimed when the object is no longer accessible (automatic garbage collection).

4/23/00

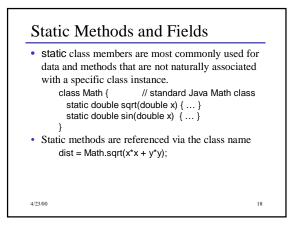
4/23/00

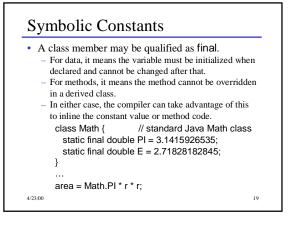
Constructors Constructor(s) can be provided to initialize objects when they are created. Constructors can be overloaded and can call other constructors. class Blob { int val; // constructors Blob (int initial) { val = initial; } Blob () { this(17); } } }

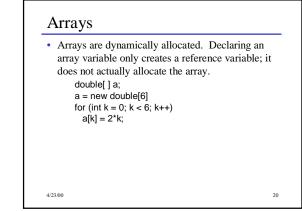
4/23/00

17

13





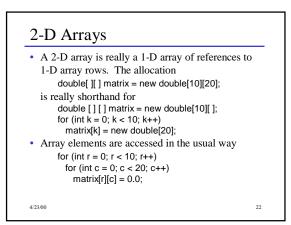


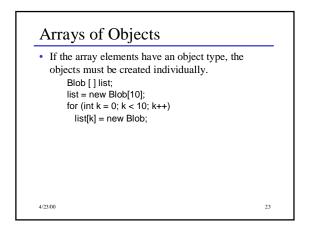
Array Notess

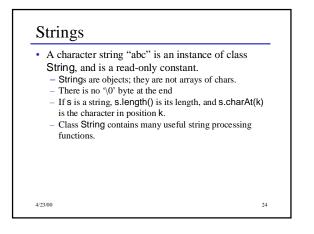
- Arrays are 0-origin, as in C/C++
 Arrays are also objects, with one constant member
- If a is an array, a.length is its length
- An IndexOutOfBoundsException is thrown if a subscript is < 0 or >= the array length.
- The brackets indicating an array type may also appear after the variable name, as in C/C++ int a[] = new int[100];

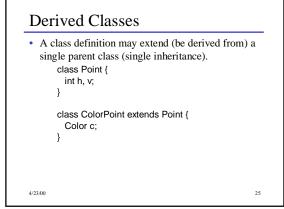
21

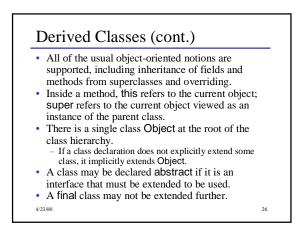
4/23/00

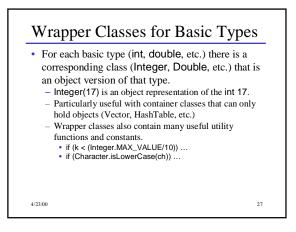


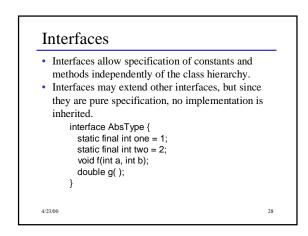


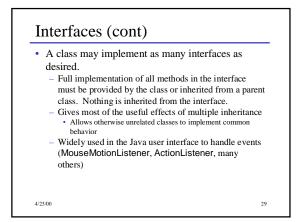


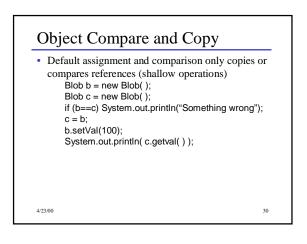








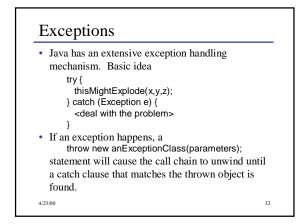


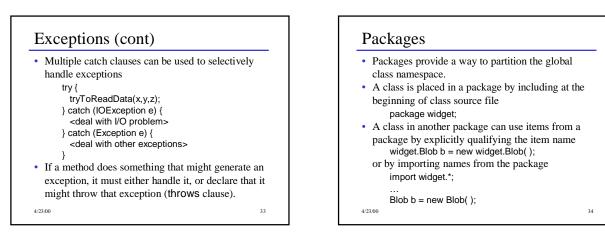


Deep Compare and Copy

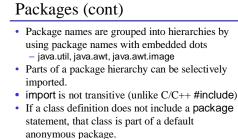
- · All classes inherit equals and clone from Object
 - Default versions do a shallow compare/copy
 - Override if a deep compare/copy is desired
 - To override clone, a class must also implement the Cloneable interface.
- Intended meaning of a.equals(b) is that a and b are "equal" in whatever sense is appropriate for the class of a and b.
- b.clone should create a new "copy" of b and return a reference to it.

4/23/00





31



- Useful for small projects

4/23/00

35

Streams

- Stream = flow of data (bytes or characters)
- Can be associated with files, communication links, keyboard/screen/printer
- Many stream classes; most are designed to be used as wrappers that accept data and transform or filter it before passing it along
- Java 1.0: Byte streams with a few wrappers to handle ASCII text
- Java 1.1: Added text stream classes to handle Unicode text properly

4/23/00

Stream Classes (1)

- InputStream/OutputStream abstract classes defining basic raw byte stream operations
- Reader/Writer abstract classes defining basic text stream operations
- All Java stream classes are built on top of these
- InputStreamReader/OutputStreamWriter basic conversion between bytes and characters (in both directions)

4/23/00

Stream Classes (2)

4/23/00

- BufferedInputStream/ BufferedOutputStream
 BufferedReader/BufferedWriter versions of streams that add buffering and additional input/output methods
- **PrintWriter** Text stream with methods for printing **String**s and primitive types as text output.

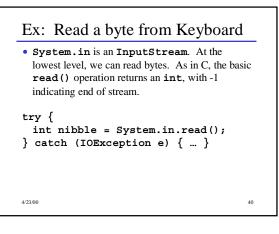
38

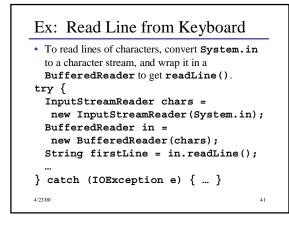
Stream Classes (3)

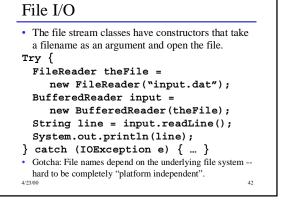
- DataInputStream/DataOutputStream -Filter streams that can read/write simple types including String and primitive numeric types as binary byte streams.
- FileInputStream/FileOutputStream FileReader/FileWriter - byte and text streams that read and write from/to the local file system.

4/23/00

39







Design Goals

- Support secure, high-performance, robust applications running as-is on multiple platforms and over networks
- "Architecture-neutral", portable, allow dynamic updates and adapt to new environments

- Look enough like C++ for programmer comfort
- Object-oriented programming
- Concurrency (multithreading)
- Simplicity