A crash course in Java

William Pugh
Univ. of Maryland, College Park

Hot linked PDF version available at:
`http://www.cs.umd.edu/~pugh/java/crashCourse`

Crash Course in Java

Roadmap

- **Day 1**
  - **What makes Java different (11-noon)**
  - **Basics (12:30 - 1:20)**
  - **Object Oriented Programming in Java (1:30 - 2:30)**
  - **Applications, Applets and Graphics (3-3:50)**
  - **Java Programming environments (4 - 5)**
- **Day 2**
  - **Exceptions and inner classes (11-noon)**
  - **Multithreading and synchronization (12:30 - 1:20)**
  - **GUI events and the Abstract Windowing Toolkit (1:30 - 2:30)**
  - **Libraries (3-3:50)**
  - **Advanced capabilities and libraries (4 - 5)**

Some notes about these slides/handouts

- **Some pages are left blank intentionally**
- **so that the first slide for each session starts at the top of a page of handouts**
- **In some PDF versions of these slides and handouts**
  - **printing is disabled**
- **People taking the course from me can get a version of the handouts that can be printed**
  - **password protected**

What Makes Java Different

Day 1, Session 1

What makes Java Different

- **Java is specified**
- **KISS principle applied**
- **Semantics are architecture insensitive**
- **Safe/Secure**
- **A modern programming language**
- **Fewer bugs?**
- **Libraries Galore!**
- **Speed**
- **Versions**
- **The Hype**
- **Microsoft's J++ vs. Java**

Java is specified

- **Pascal/C/C++ isn't**
  - `1000*1000`
  - `(-5)/10`
  - `int a[10]; for(int i=0;i<=10;i++) a[i] = 0;`
  - `delete p; q = new foo(); x = p->key; p->key = 0;`
  - `*(int *)(random()) = 0`
- **The Java specification (is intended) to completely specify the behavior of all programs**
  - **Not just "correct" programs**
  - **Caveat - multi-threading, random numbers, ...**
    - **specified but has multiple valid implementations**
  - **All run-time errors must be caught**
  - **Can make promises about what might happen**

KISS principle applied

- **Many useful features were removed from C++**
  - **Makes language easier to learn and implement**
  - **operator overloading**
  - **user-definable coercions**
  - **templates**
  - **multiple inheritance**
    - **multiple supertypes still allowed**
  - **structs/unions**
  - **unsigned integers**
  - **stand-alone functions**
- **Not essential**

Semantics are architecture insensitive

- **Not sensitive to:**
  - **size of machine word**
  - **floating point format (must use IEEE 754)**
  - **Big-endian/little-endian**
- **Compiled to machine-independent byte-code**
- **Many C/C++ programs break**
  - **when moved to machine with**
    - **different word size**
    - **different endian**

Safe/Secure

- **Can strictly limit access of a chunk of code (relies on language being specified, even for buggy programs)**
  - **Default behavior for untrusted code:**
    - **Can't access files**
    - **Network connections are restricted**
- **Can verify compiled codes!**
- **Denial of service attacks possible**
  - **and hard to prevent**
- **Security bugs possible**
- **Java is one of the smaller security risks on the net**
  - **Downloadable executables**
- **Security Risks**

Security Risks

- **If you run an program in an insecure mode**
  - **It can do anything you can do**
  - **It can set up a spy to watch what you do**
- **This includes**
  - **Erase your hard disk**
  - **Shut down your computer**
  - **Infect you with a virus**
  - **Make your Internet connection dial long distance**
  - **Add some Quicken wire transfers**
- **All C/C++ programs run in an insecure mode**
- **Signed code -- A solution?**

Signed code -- A solution?

- **Provides "proof" of who wrote the code**
  - **You might trust big companies**
  - **Allow you to track down perpetrators**
- **Can be signed by third parties**
- **If a web page erases your hard disk**
  - **allows you to easily determine who did it**
  - **but subtle attacks might be hard to catch**
- **No protection against bugs**
  - **or malicious exploitation of bugs**
- **Active-X and Java code can be signed**
- **Privileges bestowed to signed code**

Privileges bestowed to signed code

- **You can set policies about which signatures give what privileges**
- **In Active-X, all or nothing**
- **In Java**
  - **version 1.1 - applet sandbox or full power**
  - **version 1.2 - finer control**

A modern programming language

- **Includes many features that PL researchers have been advocating for years (but never caught on in mass-market)**
  - **strong type system**
  - **multi-threading and synchronization**
  - **garbage collection**
  - **exceptions**
  - **class Class**
  - **class Object**
- **Not an embarrassment to academic CS**
- **Adapts ideas from: C++, Smalltalk, Lisp, Modula-3, Objective-C**

Fewer bugs?

- **Many bugs are memory management bugs**
- **Pointers also cause problems**
- **No guarantee that shipping Java programs won't hit Exceptions/Errors**
  - **But the bugs won't propagate far**

Libraries Galore!

- **Java has a huge collection of libraries**
  - **Utilities (collection classes, Zip files, Internationalization)**
  - **Graphics/Media (2D/3D, Sound, Video)**
  - **Graphical User Interfaces**
  - **Networking (sockets, URL's, RMI, CORBA)**
  - **Threads**
  - **Databases**
  - **Cryptography/Security**
- **Increasing in each version (1.0 $\rightarrow$ 1.1 $\rightarrow$ 1.2)**
- **No other programming environments**
  - **with libraries this complete**
  - **cross-platform**
- **Huge improvement in programmer productivity**

Speed

- **Many JVM's are slow, but situation improving**
- **JVM's that do Just-in-time compilation**
- **Native code compilers**
  - **need to allow for dynamically loaded code**
- **Byte code optimizers, shrinkers and obsfucators**
- **Sun's Hotspot JVM**
- **How bad is it really?**

How bad is it really?

- **Prime number sieve - primes less than 100,000**
  - **Sun Solaris JDK 1.1.6 — 70 seconds**
  - **Sun Solaris JDK 1.2beta3/JIT — 27 seconds**
  - **Sun Solaris gcc -O4 — 21 seconds**
- **Developers use a different coding style for Java**
  - **Lots of little methods/objects, run-time type dependent stuff**
  - **This is a good thing; better programmer productivity?**
  - **But makes it hard to generate efficient code**

Versions

- **Version 1.0.2 - First stable version**
  - **Implemented in 3.x and 4.0 Netscape and IE**
- **Version 1.1.x - Java 97**
  - **Significant changes to GUI event model**
  - **Lots of new features**
  - **Available in updates to Netscape and IE**
  - **This talk assumes using at least Java 1.1**
- **Java Plug-in**
- **Version 1.2**
  - **1.2β4 released early July, 1998**
  - **1.2.0 official release scheduled Sept 21st, 1998**
- **Version 1.2 / Hotspot JVM**
  - **Beta 3Q 1998**
  - **Official release end of 1998**

The Hype

- **Cover of Businessweek !?!?**
- **Incredibly important to where Java is today**
  - **good tools**
  - **wide availability of tools and support**
  - **lots of libraries**
  - **excessive hype**
  - **overhype backlash**
- **C++ was born in the early 80's**
  - **took a decade to mature**
- **The downside**
  - **Hasty decisions have been cast in stone**
  - **A number of poor designs exist in the libraries**
    - **difficult to fix without breaking code**
  - **Religion, heat and flames**

Microsoft's J++ vs. Java

- **Bad blood between Sun and Microsoft over Java**
- **Microsoft's viewpoint**
- **Microsoft's changes to core Java functionality**

Bad blood between Sun and Microsoft over Java

- **Sun's idea is that Java allows you to write software not dependent on a particular operating system or processor**
- **Obviously not in Microsoft's interest**
- **There is a lawsuit/countersuit between Sun and Microsoft over Java**
- **From NY Times article, May 25, 1998**
  - **"necessary to fundamentally blunt Java momentum" in order "to protect our core asset, Windows" - Paul Maritz, a Microsoft group vice president**
  - **"Strategic Objective: kill cross-platform Java by growing the polluted Java market." - internal Microsoft planning document**

Microsoft's viewpoint

- **Java is a good programming language**
- **Our implementation is the fastest**
- **Our implementation is more compatible than Netscape's**
- **J++ allows/encourages you write Java programs exploit Windows-only features for better performance**
  - **fairly clear about when you use Windows-specific features**
  - **(but see next slide)**
- **Microsoft doesn't promise to track all of Sun's changes to Java**
  - **Java 1.2 changes (New security model, Swing, collections, ...)**
  - **Remote Method Invocation**

Microsoft's changes to core Java functionality

- **Microsoft has made minor changes to core packages such as `java.lang`**
  - **Some changes are not documented**
  - **Some changes expose private variables or Windows-specific features**
    - **Bad API/Programming style**
    - **Understandable - allows more efficient interfaces**
  - **Some changes are incomprehensible**
    - **e.g., leaving off a 3 line method**
    - **Either sloppy or malicious**
- **Unlikely to surprise developers**

http://www.cs.umd.edu/~pugh/java/crashCourse

A crash course in Java

Basics

Day 1, session 2

Basics

- **Mostly C/C++ syntax: statements**
- **Mostly C/C++ syntax: expressions**
- **Hello World example**
- **Naming conventions**
- **Values**
- **Object operations**
- **Special Objects**
- **Object/memory allocation**
- **Garbage Collection**
- **Other notes**
- **What is missing?**

Mostly C/C++ syntax: statements

- **Empty statement, expression statement**
- **blocks { ... }**
- **if, switch, while, do-while, for**
- **break, continue, return**
  - **any statement can be labeled**
  - **break and continue can specify a label**
  - **continue must specify a loop label**
- **throw and try-catch-finally**
- **synchronized**
- **No goto**

Mostly C/C++ syntax: expressions

- **Standard math operators: +, -, *, /, %**
- **Bit operations: &, |, ^, ~, <<,>>, >>>**
- **Update operators: =, +=, -=, *=, /=, %=, ...**
- **Relational operations: <, <=, ==,>=,>, !=**
- **Boolean operations: &&, ||, !**
- **Conditional expressions: b ? e1 : e2**
- **Select methods/variables/class/subpackage: .**
- **Class operators: new, instanceof, (Class)**
- **No pointer operations: *, &, ->**

Hello World example

```
public class HelloWorldApplication {
    public static void main(String [] args) {
        if (args.length == 1)
            System.out.println("Hello " + args[0]);
        else System.out.println("Hello World");
        }
    }
```

Naming conventions

- **Classes/Interfaces start with a capital letter**
- **packages/methods/variables start with a lowercase letter**
- **ForMultipleWords, capitalizeTheFirstLetterOfEachWord**
- **Underscores_discouraged**
- **CONSTANTS are in all uppercase**

Values

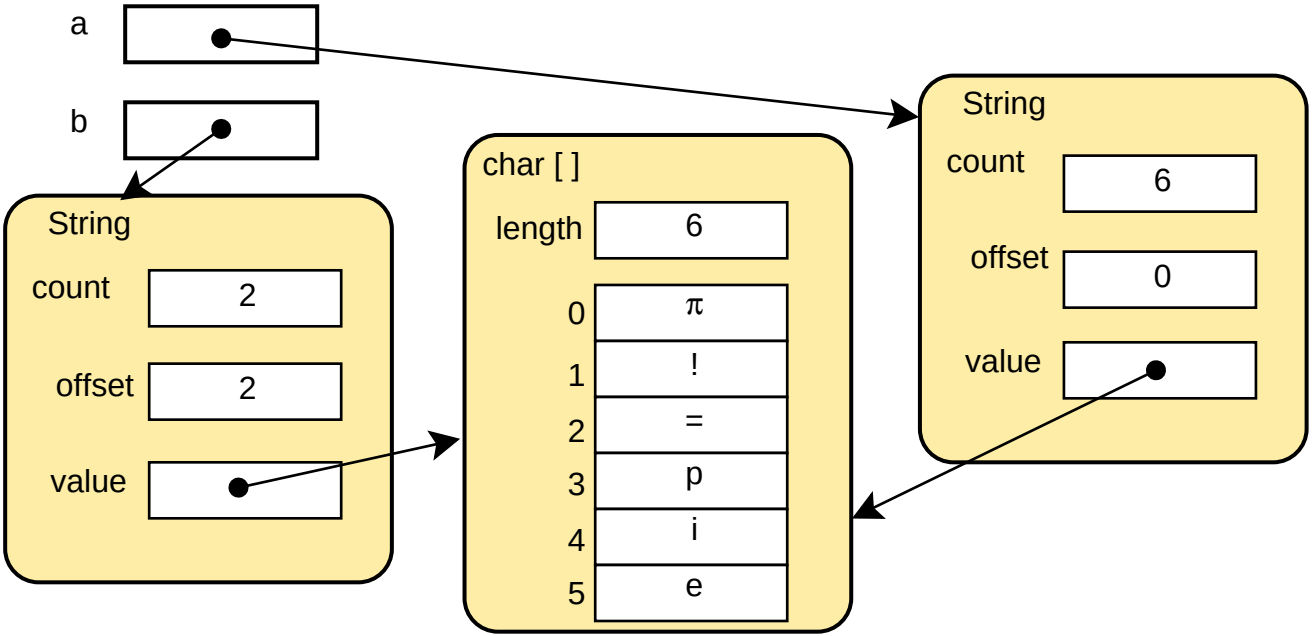
- **Object reference: null or a reference to an object**
- **boolean (Not a number or pointer/reference)**
- **char (UNICODE; 16 bits, almost a unsigned int)**
- **byte (8 bits signed)**
- **short (16 bits signed)**
- **int (32 bits signed)**
- **long (64 bits signed)**
- **float (32 bits IEEE 754)**
- **double (64 bits IEEE 754)**
- **Objects and References**

Objects and References

- **All objects are allocated on the heap**
- **No object can contain another object**
- **All class variables/fields are references to an object**
  - **A reference is almost like a C/C++ pointer, except**
    - **Can only point to start of heap allocated object**
    - **No pointer arithmetic**
    - **Use . rather than -> to access fields/methods**
- **String Example**

String Example

```
String a = "π!=pie";
String b = a.substring(2,4);
```



Object operations

- = assignment
  - For object references: copies reference, not object
- == equality test
  - For object references: true if references to same object
- foo.equals(bar)
  - By default, same as ==, but can/should be overridden
- foo.toString()
  - Returns String representation, can/should be overridden
- More Object operations

More Object operations

- `foo.clone()`
  - Returns a shallow copy of `foo` (not supported on all Objects)
- `foo.getClass()`
  - Returns class of `foo` (result is of type `Class`)
- `foo instanceof Bar`
  - true if objected referenced by `foo` is a subtype of class `Bar`
- `(Bar) foo`
  - Run-time exception if the object referenced by `foo` is not a member of a subclass of `Bar`
  - Compile-time error if `Bar` is not a subtype of `foo` (i.e., if it always throws an exception)
  - Doesn't transform anything just lets us treat the result as if it were of type `Bar`

Special Objects

- **Arrays**
- **String**

Arrays

- **Are a special kind of object (with lots of syntactic sugar)**
- **Can declare arrays of any type**
- **Arrays have one instance variable: length**
- **they also have contents indexed with a subscript of 0 ... length-1**
- **Can be initialized using {val_0, val_1, ..., val_n} notation**
  - **Initializing huge arrays this way is inefficient**
- **Array declarations**

Array declarations

- **A little surprising for C/C++ programmers**
- **`int[] A` and `int A[]` have identical semantics**
  - **declares `A` to be a variable that contains a reference to an array of int's**
- **`int[] A[], B;`**
  - **`A` is a ref to an array of ref's array of int's**
  - **`B` is a ref to an array of int's**
- **None of these allocate an array**
- **`A = new int [10]` allocates an array of 10 int's and makes `A` be a reference to it**
- **Array example**

Array example

```
int[] array1 = {1,3,5};
int[][] a = new int[10][3];
// a.length == 10
// a[7].length == 3

a[5][2] = 42;
a[9] = array1;
// a[9][2] == 5

// Use of array initializers
int[][] twoD = {{1,2,3},{4,5},{6}};
Object [] args = {"one", "two", a };
main(new String [] {"a", "b", "c"});
```

String

- A class for representing non-mutable strings
- “string constants” in program are converted into a `String`
- `+` does string concatenation
- In some contexts, objects are automatically converted to `String` type
- More about strings later...

```
public static void printArray(Object [] a) {
    for(int i=0; i < a.length; i++)
        System.out.println("a[" + i + "] = " + a[i]);
    }
```

Object/memory allocation

- **The only way/time an object gets allocated is:**
  - **by executing new**
    - **One object per invocation of new**
  - **by having a array constant (e.g., `{5, -5, 42}`)**
  - **having a string constant (e.g., `"Hello World!"`)**
  - **Declaring a reference variable doesn't allocate an object**
  - **Allocating an array doesn't automatically allocate the contents of the array**
  - **multi-array creation `int [][] a = new int[10][10];`**
    - **Equivalent to (but faster than):**

```
int [][]a = new int[10];
for(int i = 0; i < 10; i++) a[i] = new int[10];
```

- **No explicit deallocate is required/allowed**

Garbage Collection

- **Java uses garbage collection to find objects that cannot be referenced**
  - **(e.g., do not have any pointers to them)**
- **Garbage collection not currently a major bottleneck**
  - **Not as fast as it should be**
  - **Faster Garbage Collectors coming**

Other notes

- **Forward references resolved automatically**
  - **Can refer to method/variable defined later in class**
- **All integer math performed using int's or longs**
  - **Problems for unsigned shifts of shorts/bytes**
- **Integer division by zero raises an exception**
- **Integer overflow is handled by dropping extra bits**
- **Floating point errors create special values (NaN, POSITIVE_INFINITY, ...)**
- **Separate name spaces for methods, classes, variables, ...**
  - **Can produce confusing error messages**

What is missing?

- **No preprocessor (#include, #define, #ifdef, ...)**
- **No struct's or union's**
- **No enumerated types**
- **No bit-fields**
- **No variable-length argument lists**
- **Multiple inheritance**
- **Operator overloading**
- **Templates / Parameterized types**
  - **Maybe in 1.3 / 2.0**
  - **3 papers at OOPSLA98, some with Sun co-authors**
  - **Likely to require no changes to VM**

http://www.cs.umd.edu/~pugh/java/crashCourse

Object Oriented Programming

Day 1, session 3

Objects, Classes and Interfaces

- **Java Objects, constructors, instance variables and methods**
- **Superclasses and Interfaces**
- **public/protected/private methods**
- **class methods and variables**
- **final methods**

Classes

- **Each object is an instance of a class**
  - **An array is an object**
- **Each class is represented by a class object**
  - **(of type `Class`)**
- **Each class extends one superclass**
  - **(`Object` if not specified)**
  - **except class `Object`, which has no superclass**

More about Classes

- **Each class has an associated set of methods and fields/variables**
  - **Variables hold primitive values or object references**
- **Use '.' to access object fields**
  - **variables and methods**
  - **e.g., `x.y(a.b)`**
- **Most methods are invoked using C++ virtual method semantics**
  - **except static, private and final methods**

Class Modifiers

- `public` - class is visible outside package
- `final` - No other class can extend this class
- `abstract` - no instances of this class can be created
  - instances of extensions of it can

class Complex - a toy example

```
public class Complex {
  double r,i;
  Complex (double r, double i) {
     this.r = r;
     this.i = i;
     }
  Complex plus(Complex that) {
     return new Complex(
               r + that.r,
               i + that.i);
     }
  public String toString() {
     return “(“ + r
          + “,” + i + “)”;
     }
```

```
public static void main(String [] args)
  Complex a = new Complex (5.5,9.2);
  Complex b = new Complex (2.3,-5.1);
  Complex c;
  c = a.plus(b);
  System.out.println(“a = “ + a);
  System.out.println(“b = “ + b);
  System.out.println(“c = “ + c);
  }
}

a = (5.5,9.2)
b = (2.3,-5.1)
c = (7.8,4.1)
```

Details

- **You can overload method names**
  - **The method invoked is determined by both the name of the method**
  - **and the types of the parameters**
  - **resolved at compile time, based on compile-time types**
- **You can override methods: define method that is also defined by a superclass**
  - **arguments and result types must be identical**
  - **resolved at run-time, based on object method is invoked on**
- **`this` refers to the object method is invoked on**
- **`super` refers to same object as `this`**
  - **but used to access method/variables for superclass**

Methods

- **Methods are operations supported by an object/class**
- **Can be declared in both classes and interfaces**
- **Can only be implemented in classes**
- **All methods must have a return type**
  - **except constructors**
  - **void can be used only as a return type**
- **references to arrays or objects can be returned**
- **Method declaration syntax:**

```
modifiers returnType methodName ( params ) {
    [methodBody]
    }
```

Instance-Variable and Method Modifiers

- **Visibility:**
  - `public` - visible everywhere
  - `protected` - visible within same package or in subclasses
  - default (package) - visible within same package
  - `private` - visible only within this class
- `static` - a class method or variable

Instance Variable Modifiers

- `transient` - not stored when serialized
- `volatile` - never assume that the variable hasn't changed since the last time you looked at it
  - might be modified by another thread that doesn't have a lock on the object
- `final` - can't be changed, must be initialized in definition or in constructor

Method Modifiers

- `abstract` - no implementation provided
  - class must be abstract
- `final` - this method cannot be overridden
  - useful for security
  - allows compiler to inline class
- `native` - implemented in some other language
- `synchronized`
  - locks object before method is executed
  - lock released after method finishes

Method Arguments

- **Only pass-by-value**
  - **But object parameters are references to heap objects that can be changed**
- **Only arguments are used to distinguish methods**
- **Syntax same as C/C++:**

```
String print_sum (int x, int y) {
    return (“Result is: “ + (x+y));
    }
```

Overriding

- **Methods with same name and argument types in a child class override the method in the parent class**
- **You can override/hide variables**
  - **Both variables will exist**
  - **You don't want to do this**

```
class Parent {
    int cost;
    void add (int x) {
      cost += x;
      }
    }
class Child extends Parent {
    void add (int x) {
      if (x > 0) cost += x;
      }
    }
```

Overloading

- **Methods with the same name, but different parameters, either count or type are overloaded:**

```
class Parent {
   int cost;
   void add (int x) {
          cost += x;
          }
     }
class Child extends Parent {
     void add (String s) throws NumberFormatException {
          cost += Integer.parseInt(s);
     }
}
```

Dynamic method dispatch

- **If you have a ref a of type A to an object that is actually of type B (a subclass of A)**
  - **instance methods invoked on a will get the methods for class B (i.e., C++ virtual functions)**
  - **class methods invoked on a will get the methods for class A**
    - **invoking class methods on objects discouraged**
- **Simple Dynamic Dispatch example**
- **Detailed Example**

Simple Dynamic Dispatch example

```
class A {
    String f() {return "A.f() "; }
    static String g() {return "A.g() "; }
    }

public class B extends A {
    String f() {return "B.f() "; }
    static String g() { return "B.g() "; }
    public static void main(String args[]) {
        A a = new B();
        B b = new B();
        System.out.println(a.f() + a.g()
            + b.f() + b.g());
        }
    }
```

java B generates:

```
B.f() A.g() B.f() B.g()
```

Detailed Example

- **Shows**
  - **polymorphism for both method receiver and arguments**
  - **static vs instance methods**
  - **overriding instance variables**
- **Source**
- **Invocation and results**
- **What to notice**

Source

```
class A {
        String f(A x) { return "A.f(A) "; }
        String f(B x) { return "A.f(B) "; }
  static String g(A x) { return "A.g(A) "; }
  static String g(B x) { return "A.g(B) "; }
        String h = "A.h";
        String getH() { return "A.getH():" + h; }
  }

class B extends A {
        String f(A x) { return "B.f(A)/" + super.f(x); }
        String f(B x) { return "B.f(B)/" + super.f(x); }
  static String g(A x) { return "B.g(A) "; }
  static String g(B x) { return "B.g(B) "; }
        String h = "B.h";
        String getH() { return "B.getH():"
                                  + h + "/" + super.h; }
  }
```

Invocation and results

```
A a = new A(); A ab = new B(); B b = new B();

System.out.println( a.f(a) + a.f(ab) + a.f(b) );
System.out.println( ab.f(a) + ab.f(ab) + ab.f(b) );
System.out.println( b.f(a) + b.f(ab) + b.f(b) );
System.out.println();
//
// A.f(A) A.f(A) A.f(B)
// B.f(A)/A.f(A) B.f(A)/A.f(A) B.f(B)/A.f(B)
// B.f(A)/A.f(A) B.f(A)/A.f(A) B.f(B)/A.f(B)

System.out.println( a.g(a) + a.g(ab) + a.g(b) );
System.out.println( ab.g(a) + ab.g(ab) + ab.g(b) );
System.out.println( b.g(a) + b.g(ab) + b.g(b) );
System.out.println();
//
// A.g(A) A.g(A) A.g(B)
// A.g(A) A.g(A) A.g(B)
// B.g(A) B.g(A) B.g(B)


System.out.println( a.h   + "  " + a.getH());
System.out.println( ab.h  + "  " + ab.getH());
System.out.println( b.h   + "  " + b.getH());
//
// A.h  A.getH():A.h
// A.h  B.getH():B.h/A.h
// B.h  B.getH():B.h/A.h
```

What to notice

- Invoking `ab.f(ab)` invokes `B.f(A)`
  - Run-time type of object method is invoked on
  - Compile-time type of arguments
- `ab.h` gives the `A` version of `h`
- `ab.getH()`
  - `B.getH()` method invoked
  - In `B.getH()`, `h` gives `B` version of `h`
- Use of `super` in class `B` to reach `A` version of methods/variables
- `super` not allowed in static methods

Constructors

- **Declaration syntax a little strange (but same as C++):**
  - **No return type specified**
  - **“method” name same as class**
- **A class can have several Constructors**
  - **with different arguments**
- **The first statement can/should be this(args) or super(args)**
  - **If omitted, super() is used**
  - **Must be the very first thing, even before variable declarations**
- **not used for type conversions or assignments**
  - **as in C++**
- **void constructor generated if no constructors supplied**

Static components of a class

- ***Static*** **components belong to the class**
  - **Static variables are allocated once (regardless of the number of instances)**
  - **Static methods are not specific to any instance of a class and may not refer to `this` or `super`**
- **You can reference class variables and methods through either the class name or an object reference**
  - **I strongly discourage referencing them via object references;**
    - **There are big differences between instance and class variables/methods**

Interfaces

- **An interface is just an object type; no associated code or instance variables**
  - **describes methods supported by interface**
- **A class can “implement” (be a subtype of) any number of Interfaces**
- **May have final static variables**
  - **Way to define a set of constants**

Interface example

```
public interface Comparable {
    public int compareTo(Object o)
}
public class Util {
    public static void sort(Comparable []) {...}
}
public class Choices implements Comparable {
     public int compareTo(Object o) {
         return ...;
         }
}
...
    Choices [] options = ...;
    Util.sort(options);
...
```

No multiple inheritance

- **A class type can be a subtype of many other types (implements)**
- **Can only inherit method implementations from one superclass (extends)**
- **Not a significant omission (in my opinion)**
- **multiple inheritance is rarely or never necessary or well-used**
  - **“The Case against Multiple Inheritance in C++”, T.A. Cargil, *The Evolution of C++***
- **Substantially complicates implementation**

Garbage Collection

- **Objects that are no longer accessible can be garbage collected**
- **Sun's Java implements a background GC thread**
  - **needs an idle period to work**
  - **System.getRuntime.gc() forces a GC**
- **method void finalize() is called when an object is unreachable**
- **Garbage collection is not a major bottleneck**
  - **but isn't as fast as it could/should be**
  - **malloc/free isn't fast either**
  - **Faster garbage collectors are coming**

Class Objects

- For each class, there is an object of type `Class`
- Describes the class as a whole
  - used extensively in Reflection package
- `Class.forName("MyClass")`
  - returns the class object for `MyClass`
  - will load `MyClass` if needed
- `Class.forName("MyClass").newInstance()`
  - will create a new instance of `MyClass`
- `MyClass.class` will also give the `Class` object for `MyClass`

Types

- **A type describes a set of values that can be:**
  - **Held in a variable**
  - **Returned by an expression**
- **Types include:**
  - **Primitive types: boolean, char, short, int, ...**
  - **Reference types:**
    - **Class types**
    - **Array types**
    - **Interface types**

Class types

- **Using the name of a class as a type means a reference to instance of that class or a subclass is a permitted value**
  - **A subclass has all the fields of its superclass**
  - **A subclass has all the methods of its superclass**
- **Might also be `null`**

Array types

- If `S` is a subtype of `T`
  - `S[]` is a subtype of `T[]`
  - should you be surprised?
- `Object[]` is a supertype of all arrays of reference types
- A store into an array generates a run-time check that the type being stored is a subtype of the actual type of the array elements
- Performance penalty?
- Similar (and much worse) problems in C++

Object [ ]

```
public class TestArrayTypes {
    public static void reverseArray(Object [] A) {
        for(int i=0,j=A.length-1; i<j; i++,j--) {
            Object tmp = A[i];
            A[i] = A[j];
            A[j] = tmp;
            }
        }
    public static void main(String [] args) {
        reverseArray(args);
        for(int i=0; i < A.length; i++)
            System.out.println(args[i]);
        }
     }
```

Interface types

- **Using the name of an interface as a type means**
  - **a reference to any instance of a class which implements that interface is a permitted value**
  - **might also be `null`**
- **Object referenced is guaranteed to support all the methods of the interface**
  - **invoking a method on an interface might be a little less efficient**

Object Obligations

- **These operations have default implementations**
  - **which may not be the one you want**

```
public boolean equals(Object that) { ... }
  // default is return this == that
public String toString() { ... }
  // returns print representation
public int hashKey() { ... }
  // key for object
  // important that a.equals(b)
  //    implies a.hashKey() == b.hashKey()
public void finalize() { ... }
  //  called before Object is garbage collected
  //   default is {}
public void clone() { ... }
  // default is shallow bit-copy if implements Cloneable
  // throw CloneNotSupportedException otherwise
```

Poor man's polymorphism

- Every object is an `Object`
- An `Object[]` can hold references to any objects
- If we have a data structure `Set` that holds a set of `Object`
  - Can use it for a set of `String`
  - or a set of images
  - or a set of anything
- Java's container classes are all containers of `Object`
  - When you get a value out, have to downcast it

```
Hashtable h;
h.put("Key","Value");
String v = (String) h.get("Key");
```

http://www.cs.umd.edu/~pugh/java/crashCourse

Applications, Applets and Graphics

Day 1, session 4

Applications, Applets and Graphics

- **applications methods**
- **applet methods**
- **embedding applets in HTML**
- **making applets available over the web**
- **minimal Graphics**

Applications

- External interface is a public class
- with `public static void main(String []args)`
- `args[0]` is first argument (unlike C/C++)
- `System.out` and `System.err` are `PrintStream`'s
  - Should be `PrintWriter`'s, but would break 1.0 code
  - `System.out.print(...)` prints a string
  - `System.out.println(...)` prints a string and adds a newline
- `System.in` is an `InputStream`
  - Not quite as easy to use

Reading text input in (JDK 1.1) applications

- **Wrap `System.in` in a `InputStreamReader`**
  - **converts from bytes to characters**
- **Wrap it in a `BufferedReader`**
  - **makes it efficient (buffered)**
  - **supports `readLine()`**
- **`readLine()` returns a `String`**
  - **returns `null` if at EOF**

Example Echo Application

```
import java.io.*;
public class Echo {
    public static void main(String [] args) {
        String s;
        BufferedReader in = new BufferedReader(
                new InputStreamReader(System.in));
        int i = 1;
        try {
            while((s = in.readLine()) != null)
               System.out.println((i++) + ": " + s);
            }
        catch(IOException e) {
            System.out.println(e);
            }
        }
    }
```

Hello World as an applet

- **In the file HelloWorldApplet.html:**

```
<applet code=HelloWorldApplet width=300 height=40>
Your browser can't handle Java
</applet>
```

- **In the file HelloWorldApplet.java:**

```
public class HelloWorldApplet extends java.applet.Applet
    public void paint(java.awt.Graphics g) {
    // display "Hello World",
    // with start of baseline at 20,20
    g.drawString("Hello, World", 20, 20);
    }
}
```

class Applet

- **For programs that are downloaded and run within a WWW browser**
- **Minimal applet functions:**

```
public void init() // initialization code
public void paint(Graphics g) // draws applet window
public void destroy() // called when applet is purged
```

Applet/Embed tag

```
<applet code=className
    [codebase = URL]
    [archive = comma-seperated-list-of-jar-files]
    width=pixels height=pixels
    [alt="alternative test"]
    [name=appletInstanceName]
    [align=alignment]
    [hspace=pixels] [vspace=pixels]
   >
<param name=attributeName1 value=attributeValue1>
<param name=attributeName2 value=attributeValue2>
[HTML displayed if applet/embed not understood>]
</applet>
```

Example Applet HTML code

```
<applet code=DisplayTextApplet width=300 height=50>
<param name=message value="Crash Course">
<param name=fontName value=Dialog>
<param name=fontSize value=24>
</applet>
```

Try it

- **Hello world applet is at**
  - **http://www.cs.umd.edu/~pugh/crashCourse/HelloWorldApplet.html**

Making applets available over the web

- **Put class files in a directory on web server**
- **Put applet/embed code in HTML file**
  - **Point codebase to that directory**
  - **Specify class file containing applet class**

Graphics: A device-independent interface to graphics

```
setColor(Color c)
drawLine(int x1, int y1, int x2, int y2)
drawRect(int x, int y, int width, int height)
draw3DRect(int x, int y, int width, int height,
           boolean raised)
drawOval(int x, int y, int width, int height)
fillRect(int x, int y, int width, int height)
fillOval(int x, int y, int width, int height)
setFont(Font f)
drawString(String s, int x, int y)
```

java.awt.Font

- **Cross-platform fonts:**
  - `SansSerif, Serif, Monospaced, Dialog, DialogInput`
- **Font styles:**
  - `Font.PLAIN, Font.ITALIC, Font.BOLD, Font.ITALIC+Font.BOLD`
- **Font sizes: any point size allowed**
- **Constructor: Font(String name, int style, int size)**
- **Also: Font.decode(String description)**

java.awt.FontMetrics

- **Must get from a Graphics or Container object**
  - `FontMetrics fm = g.getFontMetrics(f)`
- `int stringWidth(String str)`
- `int getAscent()`
- `int getDescent()`
- DisplayTextApplet -- Source

java.awt.Color

- **Predefined colors: `Color.white`, `Color.red`, ...**
- **Constructors using RGB colors:**
  - `Color(int r, int g, int b) // 0 .. 255`
  - `Color(float r, float g, float b) // 0.0 .. 1.0`

Applet/Component Drawing Cycle

- **`update(Graphics g)`**
  - must put up the appropriate display on g
  - don't assume anything about what is up there already
  - might be what was draw by previous update()
  - applet might have been resized, iconized or obscured
  - Default behavior is to erase component, call paint
- **`paint(Graphics g)`**
  - must put up appropriate display on g
  - should assume blank canvas
  - called by default update() and print()
- **`repaint()` queues an update event**
  - updates events are combined when handled
  - No 1-1 correspondence between calls to repaint and update

More applet methods

- **Applet methods:**
  - `void init()` **// called once when initializing**
  - `void start()` **// called when applet becomes visible**
  - `void stop()` **// called when applet becomes invisible**
  - `void destroy()` **// called once when closing**
- **methods inherited from Panel/Container:**
  - `add(Component)`
- **methods inherited from Component:**
  - **get/set Foreground/Background/Font/Name/Size/Enabled**
  - **add/remove event listeners**
- **Why do Applet's have an `init()` method?**
- **Why do applets have a `destroy()` method?**

Why do Applet's have an init() method?

- **Couldn't I just use the constructor instead?**
- **Good question!**
  - **`init()` is very similar to constructor**
- **Answer:**
  - **But some context isn't set up until after applet is constructed**
    - **`setStub(AppletStub)` is called after construction**
  - **Questionable design, but makes it easier to write applets**
  - **Could figure out what is safe to do in constructor**
  - **but safer to just do it in `init()`**

Why do applets have a destroy() method?

- **Couldn't I just use `finalize()` instead?**
- **Good question!**
  - **Serve same purpose**
- **Answer:**
  - **Yes**
  - **But `destroy()` will be called sooner**
  - **need to depend on GC for `finalize()`**

Some bigger applets

- **Clock**
  - **Example:** http://www.cs.umd.edu/~pugh/java/crashCourse/Clock.html
  - **Source:** http://www.cs.umd.edu/~pugh/java/crashCourse/Clock.java
- **Graph Layout**
  - **Example:** http://www.cs.umd.edu/~pugh/java/crashCourse/Graph.html
  - **Source:** http://www.cs.umd.edu/~pugh/java/crashCourse/Graph.java
- **Tic-Tac-Toe**
  - **Example:** http://www.cs.umd.edu/~pugh/java/crashCourse/TicTacToe.html
  - **Source:** http://www.cs.umd.edu/~pugh/java/crashCourse/TicTacToe.java

Java programming environments

Day 1, session 5

Java programming environments

- **Situation constantly changing**
- **Sun's JDK freely available for most platforms**
- **GUI-creation tools that generate Java are here**
  - **Useful**
  - **Improving**

Classes are grouped into packages

- **For example, `java.awt.image`**
  - **avoids problems such as multiple `LinkedList` classes**
- **No semantics to having a common prefix**
  - **e.g., between `java.awt` and `java.awt.image`**
  - **but use them logically**
- **Package names are an implicit or explicit part of a class name**
  - **e.g., `java.awt.image.ColorModel`**

Imports make a package name implicit

- **If you import a class or package, you can use just the last name**
  - **allow use of ColorModel rather than java.awt.image.ColorModel**

    ```
    import java.awt.image.ColorModel;
    ```
  - **For each class C in java.awt.image, allow use of C rather than java.awt.image.C**

    ```
    import java.awt.image.*;
    ```
- **implicit at the beginning of every java file**

  ```
  import java.lang.*;
  ```
- **import never required, just allows shorter names**

Running Sun's JDK

- **`javac` - java compiler**
- **`java` - java intepreter**
- **`javap` - java class disassembler**
- **`jar` - Java archive tool**
- **`appletviewer` - Applet tester**
- **`javadoc` - java documentation tool**

javac - java compiler

- `javac` filenames
- e.g., `javac Test.java`
- `javac -depend Test.java`
  - Recompile Test.java and any out-of-date classes Test depends on
- `javac -d ~/java/classes Test.java`
  - Treat `~/java/classes` as the location on the classpath where files should go
  - If `Test.java` is in package `edu.umd.cs.pugh`
    - It will go in `~/java/classes/edu/umd/cs/pugh/Test.class`
- `javac -deprecation Test.java`
  - Give me detailed information about depreciated classed and methods

java - java intepreter

- **`java Classname arguments`**
  - **e.g., `java Test myInput`**
  - **e.g., `java edu.umd.edu.pugh.Test myInput`**

javap - java class disassembler

- **`javap Classname`**
  - **show fields and methods**
- **`javap -c Classname`**
  - **Show bytecodes for methods**
- **`javap -p Classname`**
  - **Show private methods/fields**

jar - Java archive tool

- **First letter of first argument is action: create/list/extract**
- **other letters are options:**
  - **f - get jar file name from next argument**
  - **m - when creating jar file, read manifest from file given as argument**
  - **v - verbose**
- **Examples**
  - `jar cvf test.jar *.class data`
  - `jar tvf test.jar`
  - `jar xf test.jar`
  - `jar xf test.jar Test.class`

appletviewer - Applet tester

- `appletviewer files`
- **One window per applet**
- **Other HTML ignored**
- **Can also supply URL's**

javadoc - java documentation tool

- **javadoc packagename**
  - **e.g., javadoc edu.umd.cs.pugh**
- **Looks for packagename on classpath**
- **Builds HTML documentation for package**
- **Special comments in java source files put into HTML**

What goes where

- **Each public class C must be in a file `C.java`**
- **If a class C is part of a package P**
  - **`package P;` must be the first statement in `C.java`**
  - **which must be in a directory `P`**
  - **Treats . in package name as sub-directories**
- **Reverse of domain name is reserved package name**
  - **`edu.umd.cs` is reserved for the Univ. Maryland CS department**
- **Classpath gives list of places to look for class files**
  - **both directories and jar/zip files**
  - **As of 1.1, you shouldn't set classpath to tell it where to find system files**
  - **You only need to set it for your own files**
    - **If there are part of a package**
    - **If they aren't in the current directory**

JAR files

- **Downloading 50 class files and 10 images over http is very expensive**
- **JAR files are compressed archives**
  - **extension of zip format**
- **Can hold class files, images, other files**
- **Java knows how to load JAR files over the net**
- **Java knows how to extract files from a JAR**
- **JAR can be signed**

java.lang

- **Wrapper classes**
- **class String**
- **class StringBuffer**

Wrapper classes

- **Allow you to create an Integer, Boolean, Double, ...**
  - **that is a subclass of Object**
  - **Useful/required for fully polymorphic methods**
    - **HashTable, ...**
  - **Used in reflection classes**
- **Including many utility functions**
  - **conversion to/from string**
    - **allows radix conversion (e.g., hexadecimal)**
  - **Many are static, don't involve creation of Wrapper object**
- **Number: superclass of Byte, Short, Integer, Long, Float and Double**
  - **allows conversion to any other numeric primitive type**

class String

- **Cannot be changed/updated**
- **String objects automatically created for string constants in program**
- **+ is used for concatenation (arguments converted to String)**
- **lots of methods, including...**
  - `int length()`
  - `char charAt(int pos)`
  - `int compare(String anotherString)`
  - `void getChars(int begin, int end, char[] dst, int dstBegin)`
  - `int indexOf(int ch) // why doesn't this take a char??`
  - `String toUpperCase()`

class StringBuffer

- **Can be changed**
- **Constructors**
  - `StringBuffer()`
  - `StringBuffer(String s)`
  - `StringBuffer(int initialBufferSize)`
- **lots of methods, including...**
  - `StringBuffer append(anything)`
  - `insert(int offset, String str)`
- **Used to implement `String` concatenation**

```
String s = "(X,Y) = (" + x + "," + y + ")"
// is compiled to:
String s = new StringBuffer("(X,Y) = (")
 .append(x).append(",").append(y).append(")").toString(
```

Cloneable

- class `Object` supports method `Object clone()`
  - but throws exception `CloneNotSupported`
  - unless you implement Cloneable
    - a hack
- Default implementation does a shallow/bitwise copy
- Sometimes you need to do something different
- standard version is protected
  - You can declare a public version
- result is of type `Object`
  - You'll probably have to downcast it

Java Surprises

- **You don’t ever need to use import**
- **Declaring a variable of class Foo doesn’t allocate an object of class Foo**
  - **All variables are references to heap allocated objects**
- **packages, classes, methods, fields and labels are separate name spaces**
- **you can label any statement and break out of it**
- **Hard to unload/update a class**
- **You need to give the full package and class name to java interpreter**
  - **but give the file name to the compiler**

More surprises

- **Internationalization makes things harder**
  - **Many things take more steps than they would in an English/US only system**
- **Threads may or may not be preemptive**
- **You can pass an `String[]` to a method that wants an `Object[]`**
  - **When you store into an array a type check is made**
- **You will write methods you never call**
  - **e.g., method `paint(Graphics g)` of an Applet**
- **And call methods you never wrote**
  - **e.g., method `repaint()` of an Applet**

Still More Surprises

- **Override update to eliminate animation flashing**
- **Beware of RuntimeExceptions**
  - **Watch out for broken sound**
  - **Exceptions in a thread just kill thread**
- **Watch for misspelling or using wrong types when overriding**

http://www.cs.umd.edu/~pugh/java/crashCourse

Exceptions and Inner classes

Day 2, session 1

Exceptions and Inner Classes

- **Exceptions**
  - **declaring exceptions**
  - **catching and throwing exceptions**
  - **Using `finally`**
- **Inner classes**
  - **introduced in Java 1.1**
  - **allows classes to be defined inside other classes**
  - **inner classes have access to variables of outer class**
  - **designed for creating helper objects**
    - **Listeners, Adaptors, ...**
  - **Fairly important for Java 1.1 GUI event model**

class Throwable

- **Just another class of objects**
- **Can be thrown**
- **Two subtypes**
  - **Exception**
  - **Error -- can be thrown without being declared**

Exception

- **Reasonable to catch and ignore exceptions**
- **IOException**
- **AWTException**
- **InterruptedException**
- **RuntimeException -- can be thrown without being declared**
  - **NullPointerException**
  - **IndexOutOfBoundsException**
  - **NegativeArraySizeException**

Error -- can be thrown without being declared

- **Generally unreasonable to catch and ignore an error**
- **VirtualMachineError**
  - **OutOfMemoryError**
  - **StackOverflowError**
- **LinkageError**
- **VerifyError**
- **NoClassDefFoundError**

method throws declarations

- **A method can declare the exceptions it might throw**

```
public void openNext() throws
    UnknownHostException,EmptyStackException { ... }
```

- **Must declare any exception you might throw**
  - **unless you catch them**
  - **includes exceptions thrown by methods you call**
- **C++ has run-time check that you don't throw any unexpected exceptions**
  - **better for backward compatibility**
- **Java uses a compile-time check**
  - **forces you to sometimes deal with exceptions that you know can't occur**

Creating New Exceptions

- **A user defined exception is just a class that is a subclass of Exception**

```
class MyVeryOwnException extends Exception { }
class MyClass{
        void oops() throws MyVeryOwnException {
                if (some_error_occurred){
                        throw new MyVeryOwnException();
                }
        }
}
```

Throwing an Exception/Error

- **Just create a new object of the appropriate Exception/Error type**
- **and throw it**
- **Unless a subtype of Error or RuntimeException**
  - **must declare that the method throws the exception**
- **Exceptions thrown are part of the return-type**
  - **when overriding a method in a superclass**
  - **can’t throw anything that would surprise a superclass**

Exception/Error Handling

- **Exceptions eventually get caught**
- **First catch with supertype of the exception catches it**
- **Don’t catch errors/throwable**
- **finally is always executed**

```
try { if (i == 0) return; myMethod(a[i]);
} catch (ArrayIndexOutOfBounds e){
        System.out.println("a[] out of bounds");
} catch (MyVeryOwnException e){
        System.out.println("Caught my error");
} catch (Exception e){
        System.out.println ("Caught" + e.toString());
        throw e;
} finally {
        /* stuff to do regardless of whether an */
        /* exception was thrown or return taken   */
}
```

java.lang.Throwable

- **Many objects of class Throwable have a message**
  - **specified when constructed**
  - **String getMessage() // returns msg**
- **String toString()**
- **void printStackTrace()**
- **void printStackTrace(PrintWriter s)**

Inner Classes

- **Allow a class to be defined within a class or method**
- **new class has access to all variables in scope**
- **classes can be anonymous**
- **4 Kinds of Inner Classes**
- **Lots of important details**

4 Kinds of Inner Classes

- **nested classes/interfaces**
- **Standard inner classes**
- **method classes and anonymous classes**

nested classes/interfaces

- **Not really an inner class**
  - **Not associated with an instance of the outer class**
- **Defined like a static class method/variable**
- **Can refer to all static methods/variables of outerclass**
  - **transparently**
- **Used to localize/encapsulate classes only used by this class**
  - **information hiding/packaging**
- **Used to package helper classes/interfaces**
  - **sort of a mini-package for each class**
- **Example**

Example

```
public class LinkedList {
 // Keep this private; no one else should see our implementation
 private static class Node {
      Object value; Node next;
      Node(Object v) { value=v; next=null; }
      };
 // Put this here so it is clear that this is the Transformer for LinkedLists
 public static interface Transformer { public Object transform(Object v); }
 Node head,tail;
 public void applyTransformer(Transformer t) {
    for(Node n = head; n != null; n = n.tail)
      n.value = t.transform(n.value);
    }
 public void append(Object v) {
      Node n = new Node(v);
      if (tail == null) head=n;
      else tail.next = n;
      tail = n;
      }
 }

public class getStringRep implements LinkedList.Transformer {
  public Object transform(Object o) { return o.toString(); }
  }
```

Standard inner classes

- **Defined like a class method/variable**
- **Each instance associated with an instance of the outer class**
- **If class `A` is outer class**
  - **use `A.this` to get `this` for instance of outer class**
- **Can refer to all methods/variables of outerclass**
  - **transparently**
- **Can't have any static methods/variables**
- **Example**

Example

```
public class FixedStack {
 Object array [];
 int top = 0;
 class MyEnum implements java.util.Enumerator {
    int count = top;
    public boolean hasMoreElements() { return count > 0; }
    public Object nextElement() {
        if (count == 0)
           throw new NoSuchElementException("FixedStack");
        return array[--count];
        }
    }
 public java.util.Enumerator enumerateAll() {
            return new MyEnum(); }
}
```

method classes and anonymous classes

- **Can refer to all methods/variables of outerclass**
- **Can refer to final local variables**
- **Can’t have any static methods/variables**
- **Method classes defined like a method variable**
- **Anonymous classes defined in new expression**

```
new BaseClassOrInterface() { extensions }
```

- **Method class Example**
- **Anonymous class Example**

Method class Example

```
public class FixedStack {
  Object array [];
  int top = 0;
  public java.util.Enumerator enumerateOldestK(final int k) {
   class MyEnum implements java.util.Enumerator {
     int pos = 0;
     public boolean hasMoreElements()
          { return pos < k && pos < top; }
     public Object nextElement() {
       if (!hasMoreElements())
        throw new NoSuchElementException("FixedStack");
       return array[pos++];
       }
     }

   return new MyEnum(); }
}
```

Anonymous class Example

```
public class FixedStack {
  Object array [];
  int top = 0;
  public java.util.Enumerator enumerateOldestK(final int k) {
    return  java.util.Enumerator() {
      int pos = 0;
      public boolean hasMoreElements() { return pos < k && pos < top;
      public Object nextElement() {
        if (!hasMoreElements())
         throw new NoSuchElementException("FixedStack");
        return array[pos++];
        }
     }
}
```

Lots of important details

- **If class `B` is defined inside of class `A`**
  - **A synchronized method of `B` locks `B.this`, not `A.this`**
  - **You may want to lock `A.this` for synchronization**
  - **Can have many `B`'s for each `A`**
- **Can't define constructor for anonymous inner class**
- **Inner classes are a compile-time transformation**
  - **separate class file generated for each inner class**
  - **$'s in names**

http://www.cs.umd.edu/~pugh/java/crashCourse

Multithreading and Synchronization

Day 2, session 2

Multithreading and Synchronization

- **What is a thread?**
- **Writing Multithreading code can be difficult**
- **Working with Threads**
- **Synchronization**
- **Depreciated Methods on Threads**
- **A common multithreading bug**
- **Some guidelines to simple/safe multithreaded programming**

What is a thread?

- **A thread is a program-counter and a stack**
- **All threads share the same memory space**
  - **take turns at the CPU**
- **WWW browser:**
  - **One thread handling I/O**
  - **One thread for each file being downloaded**
  - **One thread to render web page**
- **The running thread might:**
  - **Yield**
  - **Sleep**
  - **Wait for I/O or notification**
  - **Be pre-empted**
- **On multiprocessor, concurrent threads possible**

Writing Multithreading code can be difficult

- **Need to control which events can happen simultaneously**
  - **e.g., update and display method**
- **Normally covered only in OS and/or DB courses**
  - **few programmers have substantial training**
- **Can get inconsistent results or deadlock**
  - **problems often not reproducible**
- **Very easily to get multithreading, even without trying**
  - **Graphical User Interfaces (GUI's)**
  - **Remote Method Invocation**

Working with Threads

- **extending class Thread**
- **Simple thread methods**
- **Simple static thread methods**
- **interface Runnable**
- **Thread Example**
- **InterruptedException**
- **Be careful**
- **Another thread example**
- **Daemon threads**

extending class Thread

- Can build a thread by extending `java.lang.Thread`
- You must supply a `public void run()` method
- Start a thread by invoking the `start()` method
- When a thread starts, it executes `run()`
- When `run()` finished, the thread is finished/dead

Simple thread methods

- `void start()`
- `boolean isAlive()`
- `void setDaemon(boolean on)`
  - If only daemon threads are running, VM terminates
- `void setPriority(int newPriority)`
  - Thread schedule *might* respect priority
- `void join() throws InterruptedException`
  - Waits for a thread to die/finish

Simple static thread methods

- **Apply to thread invoking the method**

```
void yield()
void sleep(long millisecs)
      throws InterruptedException
Thread currentThread()
```

interface Runnable

- extending Thread means can't extend anything else
- Instead `implement Runnable`
  - Declares that an object has a `void run()` method
- Create a new `Thread`
  - giving it an object of type `Runnable`
- Constructors:

```
Thread(Runnable target)
Thread(Runnable target, String name)
```

Thread Example

```
public class ThreadDemo implements Runnable {

    public void run() {
        for (int i = 5; i> 0; i--) {
            System.out.println(i);
            try { Thread.sleep(1000); }
            catch (InterruptedException e) { };
        }
        System.out.println("exiting " + Thread.currentThread() );
    }

    public static void main(String args[])
    {
        Thread t = new Thread(new ThreadDemo(), "Demo Thread");
        System.out.println("main thread: " + Thread.currentThread());
        System.out.println("Thread created: " + t);
        t.start();
        try { Thread.sleep(3000); }
        catch (InterruptedException e){ };
        System.out.println("exiting " + Thread.currentThread() );
    }
}
```

InterruptedException

- **A number of thread methods throw this exception**
  - **Really means: wakeUpCall**
- **`interrupt()` sends a wakeUpCall to a thread**
- **won't disturb the thread if it is working**
  - **however, if the thread attempts to sleep**
  - **it will get immediately woken up**
- **will also wake up the thread if it is already asleep**
- **thrown by `sleep(), join(), wait()`**

Be careful

- **Under some implementations**
  - **a thread stuck in a loop will never yield by itself**
- **Preemptive scheduling would guarantee it**
  - **not supported on all platforms**
- **Put yield() into loops**
- **I/O has highest priority, so should be able to get time**

Another thread example

```
class UnSyncTest extends Thread {
     String msg;
     public UnSyncTest(String s) {
       msg = s;
       start();
       }

    public void run() {
       System.out.print("[" + msg);
       try { Thread.sleep(1000); }
          catch (InterruptedException e) {};
       System.out.println("]");
       }

    public static void main(String args[]) {
      new UnSyncTest("Hello");
      new UnSyncTest("UnSynchronized");
      new UnSyncTest("World");
      }
}
```

Daemon threads

- **A thread can be marked as a Daemon thread**
- **By default, acquire status of thread who spawned you**
- **When nobody running except Daemons**
  - **Execution terminates**

Synchronization

- **Locks**
- **Synchronized methods**
- **Synchronized statement**
- **Example with Synchronization**
- **Using wait and notify**
- **ProducerConsumer example**
- **A change**
- **A Better Fix**
- **Deadlock**

Locks

- All objects can be locked
- Only one thread can hold a lock on an object
  - Other threads block until they can get it
- If your thread already holds a lock on an object
  - you can lock it a second time
  - object not unlocked until both locks released
- No way to attempt getting a lock

Synchronized methods

- **A method can be synchronized**
  - **add `synchronized` before return type**
- **Obtains a lock on object referenced by `this` before starting method**
  - **releases lock when method completes**
- **A static synchronized method**
  - **locks the class object**

Synchronized statement

- `synchronized (obj) {` **block** `}`
- **Obtains a lock on obj before executing block**
- **Releases lock once block completes**
- **Provides finer grain of control**
- **Allows you to lock arguments to a method**

Example with Synchronization

```
class SyncTest extends Thread {
     String msg;
     public SyncTest(String s) {
       msg = s;
       start();
       }

    public void run() {
      synchronized (SyncTest.class) {
       System.out.print("[" + msg);
       try { Thread.sleep(1000); }
          catch (InterruptedException e) {};
       System.out.println("]");
       }
     }

    public static void main(String args[]) {
      new SyncTest("Hello");
      new SyncTest("Synchronized");
      new SyncTest("World");
      }
}
```

Using wait and notify

- `a.wait()`
  - Gives up lock(s) on `a`
  - adds thread to wait set for `a`
  - suspends thread
- `a.wait(int m)`
  - limits suspension to `m` milliseconds
- `a.notify()` resumes one thread from `a`'s wait list
  - and removes it from wait set
  - no control over which thread
- `a.notifyAll()` resumes all threads on `a`'s wait list
- resumed tasks must reacquire lock before continuing
- wait doesn't give up locks on any other objects

ProducerConsumer example

```
public class ProducerConsumer {
 private boolean ready = false;
 private Object obj;
 public ProducerConsumer() {  }
 public ProducerConsumer(Object o) {
   obj = o;
   ready = true;
 }
                        synchronized Object consume() {
                           while(!ready) wait();
                           ready=false;
                           notifyAll();
                           return obj;
                           }
                        synchronized void produce(Object o) {
                           while(ready) wait();
                           obj = o;
                           ready=true;
                           notifyAll();
                           }
                        }
```

A change

```
synchronized void produce(Object o) {
   while(ready) {
       wait();
       if (ready) notify();
       }
   obj = o;
   ready=true;
   notify();
   }
```

```
synchronized Object consume() {
   while(!ready) {
       wait();
       if (!ready) notify();
       }
   ready=false;
   notify();
   return obj;
   }
```

Bad design - no guarantee about who will get woken up

A Better Fix

```
void produce(Object o) {
   while(ready) { synchronized (empty) {
          try {empty.wait();}
          catch (InterruptedException e) {}
        }}
     obj = o;   ready=true;
     synchronized (full) {
               full.notify();
               }
     }
```

```
Object consume() {
   while(!ready) { synchronized (full) {
          try { full.wait();}
          catch (InterruptedException e) {}
          }}
     Object o = obj;   ready=false;
     synchronized (empty) {
          empty.notify();
          }
}
```

Use two objects, **`empty`** and **`full`**, to allow two different wait sets

Deadlock

- **Quite possible to create code that deadlocks**
  - **Thread 1 holds a lock on A**
  - **Thread 2 holds a lock on B**
  - **Thread 1 is trying to obtain a lock on B**
  - **Thread 2 is trying to obtain a lock on A**
  - **deadlock!**
- **Not easy to detect when deadlock has occurs**
  - **Other than by the fact that nothing is happening**

Depreciated Methods on Threads

- **The following methods are depreciated in Java 1.2**
  - **Discouraged**
  - **Will probably still work**
- **`t.stop()` -- kills thread t**
  - **causes a ThreadDeath Error to be thrown**
- **`t.suspend()` -- halts thread t**
  - **retains all locks held while suspended**
- **`t.resume()` - wakes up suspended thread t**

A common multithreading bug

- **Threads might cache values**
- **Obtaining a lock forces the thread to get fresh values**
- **Releasing a lock forces the thread to push out all pending writes**
- **volatile variables are never cached**
- **`sleep(...)` doesn't force fresh values**
- **Current compilers don't current perform these optimizations**
  - **Hotspot may**
- **Problem might also occur with multiple CPU's**
- **Example of common multithreading bug**

Example of common multithreading bug

- **From Bruce Eckel's "Thinking in Java"**
  - **mostly an excellent book**
- **Problems with this example**
  - **No way for thread to gracefully die**
  - **`runFlag` might be cached (never see changes by other threads)**
  - **`c2.t` might be cached (write never seen by other threads)**

```
while (true) {
    try {
        sleep(100);
    } catch (InterruptedException e) {};
    if (runFlag)
        c2.t.setText(Integer.toString(count++));
    }
```

Some guidelines to simple/safe multithreaded programming

- **Synchronize/lock access to shared data**
- **Don't hold a lock on more than one object at a time**
  - **could cause deadlock**
- **Hold a lock for as little a time as possible**
  - **reduces blocking**
- **While holding a lock, don't call a method you don't understand**
  - **e.g., a method provided by another client**
- **Have to go beyond this for sophisticated situations**
  - **But need to understand threading/synchronization well**
- **Recommended book for going further:**
  - ***Concurrent Programming in Java*** **by Doug Lea**

A crash course in Java

Abstract Windowing Toolkit

Day 2, session 4

Abstract Windowing Toolkit

- **The AWT is very complex (as is any GUI library)**
- **The event model was changed substantially for version 1.1**
  - ***Big* improvement**
  - **Inter-operable, but not within the same window**
- **To keep things manageable, I'll only discuss 1.1 model**
  - **Only reason to use 1.0 model is to be compatible with older browsers**

Widgets/Components

- **Container - Panel or Window**
- **Button**
- **Checkbox**
- **Choice**
- **Label**
- **List**
- **Scrollbar**
- **TextField**
- **TextArea**

Automatic Layout Managers

- **Determine position and size of components**
- **Depends on minimum, preferred and maximum size of components**
- **Allows resizing of windows**
  - **Controls where extra space goes**
- **Allows for the fact that on different platforms and in different languages**
  - **Components might have different sizes**
- **Without a layout manager, must position each component**
- **Can write your own layout manager**
- **Several layout managers provided**

Several layout managers provided

- `BorderLayout` - North/South/West/East/Center
- `FlowLayout` - Like a word processor
- `CardLayout` - Multiple layouts
  - only one of which is displayed at a moment
  - Like a tabbed layout, but no tabs
- `GridLayout` - a regular grid
  - All grid elements same size
- `GridBagLayout` -- like an HTML table
  - Components can span multiple columns/rows
  - Can control where extra space is directed
  - Very powerful and very awkward

Event Handling in version 1.1

- **Components allow you to attach listeners**
  - **Different components allow different listeners**
    - `ActionListener`
    - `TextListener`
    - `FocusListener`
    - `MouseListener`
  - **When a component gets an event, it sends the event to all attached listeners**

Example 1.1 Event handling - part 1

```
import java.awt.*;
import java.awt.event.*;

public class EventHandling {
     GUI gui = new GUI();
    void search(ActionEvent e) { System.out.println("Search: " + e); }
    void sort(ActionEvent e)   { System.out.println("Sort: " + e); }
    void check(ItemEvent e)    { System.out.println("Check: " + e); }
    void text(ActionEvent e)   { System.out.println("Text event: " + e); }
    void text(TextEvent e)     { System.out.println("Text: " + e); }
    static public void main(String args[]) {
        EventHandling app = new EventHandling();
    }
```

Example 1.1 Event handling - part 2

```
class GUI extends Frame {   // Innerclass of EventHandling
   public GUI() {
      super("EventHandling");
      setLayout(new FlowLayout());
      Button b;
      add(b = new Button("Search"));
      b.addActionListener(
         new ActionListener() {
            public void actionPerformed(ActionEvent e) { search(e); }
              });
      add(b = new Button("Sort"));
      b.addActionListener(
           new ActionListener() {
               public void actionPerformed(ActionEvent e) { sort(e); }
              });
      Checkbox cb;
      add( cb = new Checkbox("alphabetical"));
      cb.addItemListener(
            new ItemListener() {
               public void itemStateChanged(ItemEvent e) { check(e); }
                 });
```

Example 1.1 Event handling - part 3

```
Choice c;
add ( c = new Choice());
c.addItem("Red");
c.addItem("Green");
c.addItem("Blue");
c.addItemListener(
    new ItemListener() {
         public void itemStateChanged(ItemEvent e) { check(e); }
        });
TextField tf;
add(tf = new TextField(8));
tf.addActionListener(
     new ActionListener() {
          public void actionPerformed(ActionEvent e) { text(e); }
         });
tf.addTextListener(
     new TextListener() {
          public void textValueChanged(TextEvent e) { text(e); }
         });
pack(); show();
}}}
```

Removing animation flicker

- **Default `update()` method for applets**
  - **Erase to background color**
  - **call `paint()` to draw new image on clean background**
  - **Can cause flicker**
- **Eliminate flicker**
  - **Erase offscreen image**
  - **paint onto a offscreen image**
  - **copy offscreen image onto screen**
- **class FlickerFree**
- **Anyone who extends FlickerFreeApplet is flicker free**

class FlickerFree

```
public class FlickerFreeApplet extends Applet {
  private Image      offscreenImage;
  private Graphics   offscreenGraphics;
  private Dimension  offscreenDimension;
```

FlickerFreeApplet's update

```
public final void update(Graphics g)  {
     Dimension d = size();
     // warning! In 1.0, Dimension.equals is broken
     if (offscreenImage == null || !d.equals(offscreenDimension)) {
          offscreenDimension = d;
          offscreenImage =
               createImage(offscreenDimension.width,
                        offscreenDimension.height);
          offscreenGraphics = offscreenImage.getGraphics();
          };
     offscreenGraphics.setColor(getBackground());
     offscreenGraphics.fillRect(0,0,
               offscreenDimension.width,offscreenDimension.height);
     offscreenGraphics.setColor(getForeground());
     offscreenGraphics.setFont(getFont());
     paint(offscreenGraphics);
     g.drawImage(offscreenImage,0,0,this);
     }
```

http://www.cs.umd.edu/~pugh/java/crashCourse

I/O, Networking and Utility libraries

Day 2, session 5

I/O, Networking and Utility libraries

- **I/O classes**
- **URL's and Web connections**
- **Sockets**
- **java.util**
- **Other libraries**
- **Loading Resources**

I/O classes

- **File**
  - **directories: if (f.isDirectory()) System.out.println(f.list());**
  - **interface FilenameFilter -- allows selection of sublist**
- **OutputStream - byte stream going out**
- **Writer - character stream going out**
- **InputStream - byte stream coming in**
- **Reader - character stream coming in**

OutputStream - byte stream going out

- **base types**
  - **ByteArrayOutputStream**
  - **FileOutputStream - goes to file**
  - **PipedOutputStream - goes to PipedInputStream**
  - **SocketOutputStream (not public) - goes to TCP socket**
- **Filters - wrapped around an OutputStream**
  - **BufferedOutputStream**
  - **ObjectOutputStream (should implement FilterOutputStream)**

Writer - character stream going out

- **OutputStreamWriter**
  - **wrap around OutputStream to get a Writer**
  - **Takes characters, converts to bytes**
  - **Can specify encoding used to convert characters to bytes**
- **CharArrayWriter**
- **StringWriter**
- **Filters**
  - **PrintWriter - supports print, println**
  - **BufferedWriter**
- **Convenience Writers**
  - **(wraps an OutputStreamWriter around an OutputStream)**
  - **FileWriter**
  - **PipedWriter**

InputStream - byte stream coming in

- **base types**
  - **ByteArrayInputStream**
  - **FileInputStream**
  - **PipedInputStream**
  - **SocketInputStream (not public) - comes from to TCP socket**
- **Filters - wrapped around an InputStream**
  - **BufferedInputStream**
  - **PushbackInputStream**
  - **ObjectInputStream**
- **SequenceInputStream -- cat**

Reader - character stream coming in

- **InputStreamReader**
  - **Wrap around an InputStream to get a Reader**
  - **takes bytes, converts to characters**
  - **Can specify encoding used to convert bytes to characters**
- **CharArrayReader**
- **StringReader**
- **Filters**
  - **BufferedReader - efficient, supports readLine()**
    - **LineNumberReader - reports Line Numbers**
  - **PushbackReader**
- **Convenience Readers**
  - **wraps an InputStreamReader around an InputStream**
  - **FileReader**
  - **PipedReader**

URL’s and Web connections

- **Example: URLGet**
- **URL’s**
- **URLConnection**

Example: URLGet

```
import java.net.*;
import java.io.*;
public URLGet {
  public static main(String [] args) throws Exception {
     if (args.length != 1) {
          System.out.println("Please supply one URL as an argument")
          System.exit(1);
          }
     URL u = new URL(args[0]);
     BufferedReader in = new BufferedReader(
         new InputStreamReader(u.openConnection().getInputStream()))
     String s;
     while((s = in.readLine()) != null) System.out.println(s);
    }
}
```

URL's

```
URL u = new URL("http://www.cs.umd.edu:8080/index.html");
// then
    URLConnection conn = u.openConnection();
    // or
    InputStream in = u.openStream();
    // or
    Object o = u.getContents();
     // depends on finding ContentHandler
     // parses content
     // e.g., JPEG file turned into image
```

URLConnection

```
int len = conn.getContentLength();
// Number of bytes in content

long date = conn.getDate();
// Time of last modification
// Milliseconds since Epoc
// Convert to Date() for other forms

String type = conn.getContentType();
// Get Mime type of content

Object o = conn.getContent();
// finds ContentHandler to parse Contents
```

Sockets

- **Sockets are Internet's way of sending/receiving messages**
  - **everything is done via a socket**
  - **Supports**
    - **TCP sockets**
      - **guaranteed, stream based communication**
    - **UDP sockets**
      - **unguaranteed, packet based communications**
      - **also supports Multicast UDP sockets**
  - **TCP Client Socket Example**
  - **TCP Server Socket Example**

TCP Client Socket Example

```
import java.net.*;
import java.io.*;
public class SocketGet {
  public static void main(String [] args) throws Exception {
    if (args.length != 2) {
      System.out.println(
         "Please supply a hostname and port as arguments");
      System.exit(1);
      }
    Socket s = new Socket(args[0],Integer.parseInt(args[1]));
    BufferedReader in = new BufferedReader (
        new InputStreamReader (s.getInputStream());
    String m;
    while((m = in.readLine()) != null)
        System.out.println(m);
    s.close();
    }}
```

TCP Server Socket Example

```
import java.net.*;
import java.io.*;
public class SocketServe {
  public static void main(String [] args) throws Exception {
    if (args.length != 2) {
      System.out.println(
          "Please supply a port and a msg as arguments");
      System.exit(1);
      }
    ServerSocket Srv = new ServerSocket(Integer.parseInt(args[0]));
    while (true) {
       Socket s = Srv.accept();
       PrintWriter out = new PrintWriter(s.getOutputStream());
       out.println(args[1]);
       out.close();
       s.close();
       }
    }
}
```

java.util

- **Vector**
- **Dictionaries**
- **Enumerations and Bitsets**
- **Miscellaneous**
- **Java 1.2 Collection Classes**

Vector

- **A list/vector abstraction**
- **Can insert/delete/modify elements anywhere in list**
- **Can access by position**
- **Stack**
  - **An extension of Vector**
  - **Adds push, pop, peek and empty**

Dictionaries

- **Dictionary**
  - **An abstract class**
  - **Represents a key to value mapping**
- **HashTable**
  - **An implementation of Dictionary**
- **Properties**
  - **Uses HashTable**
  - **Keys and Values are Strings**
  - **Allows scoping**
  - **Can be saved to a file**

Enumerations and Bitsets

- **Enumeration**
  - **Just an Interface**
  - **Used in a number of places to return an enumeration**
    public boolean hasMoreElements()
    public Object nextElement()
- **BitSet**
  - **Provides representation of a set as a bitvector**
  - **Grows as needed**

Miscellaneous

- **Date**
  - **Not a great design**
  - **1.1 adds java.util.Calendar and java.text.DateFormat**
    - **many Date methods depreciated**
    - **Complicated due to internationalization**
      - **and bad design?**
- **Random**
- **StringTokenizer**

```
StringTokenizer tokens = new StringTokenizer(msg," ");
   while (tokens.hasMoreTokens())
       System.out.println(tokens.nextToken());
```

- **java.util.zip package**
  - **Provides ability to read/write zip archives**

Java 1.2 Collection Classes

- interface Collection
  - interface List
    - class Vector
      - class Stack
    - class ArrayList
    - class LinkedList - a doubly linked list
  - interface Set
    - class HashSet
    - interface SortedSet
      - class TreeSet
- interface Map - Dictionary like structures
  - class HashMap; replaces HashTable
  - interface SortedMap
    - class TreeMap

Other libraries

- **class java.lang.Math**
  - **abstract final class - has only static members**
  - **Includes constants E and PI**
  - **Includes static methods for trig, exponentiation, min, max, ...**
- **java.text Package**
  - **New to Java 1.1**
  - **Text formatting tools**
    - **java.text.MessageFormat provides printf/scanf functionality**
  - **Lots of facilities for internationalization**

Loading Resources

- **Can load resources from same place as class**
  - **images**
  - **Text files**
  - **Serialized Objects**
  - **local file or http connection**
  - **directory or jar file**
    - **easiest way to get data out of a jar file**
- **Code snippets**

Code snippets

- `URL u = obj.getClass().getResource("title.gif");`
  - gets URL for title.gif from the same place as the class file for obj
  - Doesn't work in Netscape
  - `u.getContent()` gets content
    - `java.awt.image.ImageProducer` for images
- `InputStream in = getClass().getResourceAsStream("data");`
  - Gives access to raw bytes
  - Works in Netscape

http://www.cs.umd.edu/~pugh/java/crashCourse

Advanced capabilities/libraries

Day 2, session 5

Advanced capabilities/libraries

- **Object Serialization**
- **Remote Method Invocation**
- **Security**
- **JavaBeans**
- **Reflection**
- **Java DataBase Connection (JDBC)**
- **Drag-n-Drop, Clipboard**
- **2D/3G graphics**

Object Serialization

- **Allows you to write/read object to/from a stream**
- **Correctly handles graphs and cycles**
- **Two ways to allow on object to be serialized**
  - **`implement Serializable` -- depend on system**
  - **`implement Externalizable` -- roll your own**
- **Version control a tricky and difficult problem**
  - **if you don't do anything, can't read previous versions**
  - **Can OK reading old versions**
    - **set serialVersionUID**

implement Serializable -- depend on system

- **Can define readObject**

```
private void readObject(ObjectInputStream in)
    throws IOException, ClassNotFoundException
```

  - Can invoke `in.defaultReadObject()`
    - restores all non-static, non-transient fields
- **Can define writeObject**

```
private void writeObject(ObjectOutputStream out)
    throws IOException, ClassNotFoundException
```

  - Can invoke `out.defaultWriteObject()`
    - saves all non-static, non-transient fields

implement Externalizable -- roll your own

- **to read an object:**

```
public void readExternal(ObjectInput in)
    throws IOException
```

- **to write an object:**

```
public void writeExternal(ObjectOutput out)
    throws IOException
```

Remote Method Invocation

- **Set up registry to allow you to locate remote objects by name**
- **Allows methods to be invoked on remote objects**
  - **Parameters and result copied by-value using serialization**
  - **Except that remote objects aren't copied**
    - **instead, a remote reference is passed**
- **Similar to CORBA, but**
  - **only works Java-to-Java**
  - **easier to use**
- **RMI Agents**

RMI Agents

- **A program using RMI can specify a codebase**
  - **URL that provides access to class files**
- **If an object x of Class Y is sent from machine A to machine B**
  - **If B can’t locate code for Class Y locally**
  - **B retrieves it from A’s codebase**

Security

- **Code can be digitally signed**
- **Determines privileges code will get**

JavaBeans

- **Use the Bean coding style, and your class is a JavaBean**
  - **use getXXX() method to get value of property XXX**
  - **use setXXX() method to set value of property XXX**
  - **Similar styles for attaching EventListeners, ...**
  - **Can also provide code that describes this info**
- **Bean development environments**
  - **Work Visually**
  - **Allow you to connect and customize Beans**
  - **Customized Beans can be serialized and saved**
- **Many environments have similar visual programming tools**
  - **But JavaBeans are very easy to create**

Reflection

- **Reflection as in looking in a mirror**
- **Allows examination of the methods supported by a class at run time**
- **allows invocation of calls you didn't know existed at compile time**
- **Useful for lots of tools:**
  - **Visual programming environments**
  - **Java Beans**
  - **Serialization**
  - **RMI**
- **Example use: InvokeMain.java**

Example use: InvokeMain.java

- **Given name of class and arguments**
- **invokes static main method with those arguments**
- **doesn't work well with programs that check for EOF of standard input**

```
Class classToInvoke = Class.forName(className);
Object[] argumentsToInvoke = new Object[1];
argumentsToInvoke[0] = args;

Method mainMethod
     = classToInvoke.getMethod("main",argsTypeForMain);
mainMethod.invoke(null, argumentsToInvoke);
```

Java DataBase Connection (JDBC)

- **Allows online connect to SQL relational database**
- **Allows full power SQL**
- **Designed to allow use in serious database applications**
- **Most database vendors are providing JDBC interfaces**

Drag-n-Drop, Clipboard

- **Allows information to be cut-and-pasted or dragged-and-dropped**
- **Data can have multiple data flavors**
  - **A graph could be supplied as**
    - **a picture**
    - **a data series**
    - **text**

2D/3G graphics

- **2D graphics package is a replacement for java.awt.Graphics**
  - **Allows more powerful operations (affine transformations, ...)**
- **3D graphics provides interface to 3D graphics system**
  - **will probably require tuned software or special hardware**