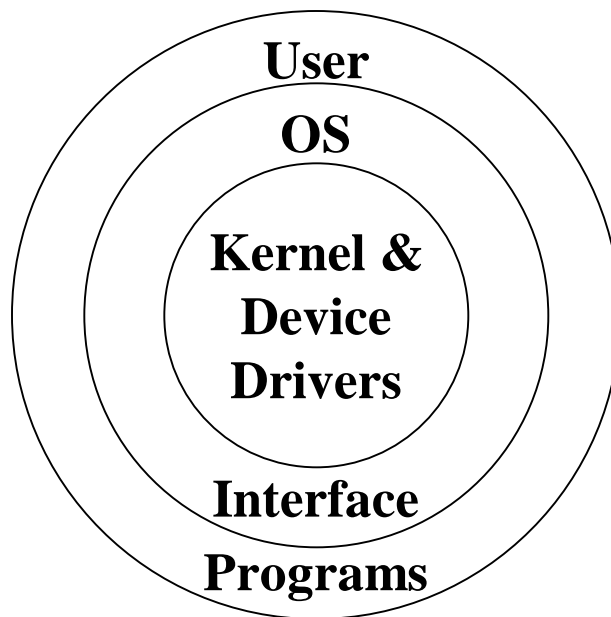
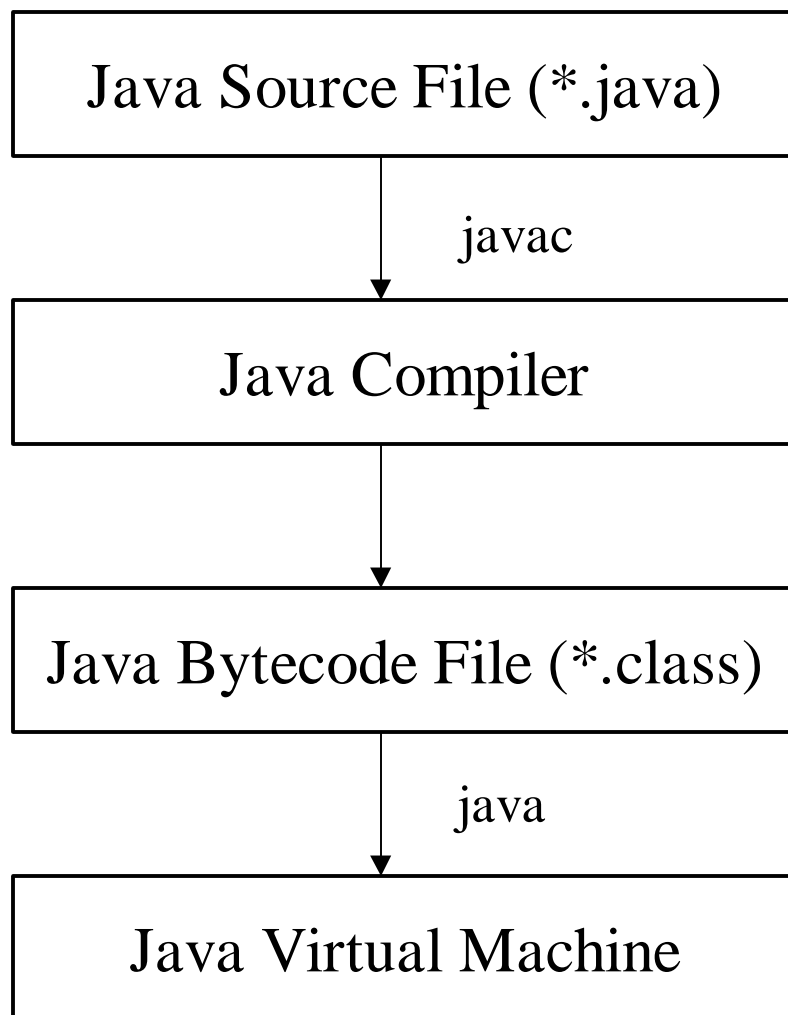


Operating Systems



A Crash Course In Java

The Java Environment



Java Features

- “Write Once, Run Anywhere”
- Portability possible because of Java virtual machine technology
 - Interpreted
 - JIT Compilers
- Many language features that we are interested in:
 - Concurrent programming
 - Support for simulating important operating system algorithms
- Similar to C++
 - Clearer semantics
 - No pointers
 - Automatic garbage collection
- Java is easy to extend
 - Corba, EC, animation, networking, ...

Java

- Java is a very powerful programming language
- We will focus on a minimal set of the language that will allow us to develop and simulate interesting operating system algorithms
- One source code file, with one or more classes
- Input and output will be character (terminal) based
 - Java has AWT and swing for fancy UI

Java Virtual Machine

- Java programs run within a Java Virtual machine
- Simulator for Java bytecode
- Provides a virtual environment to run Java programs
- Features
 - Security
 - Portability
 - Small bytecode footprint
 - Superior dynamic resource management
 - Resource location transparency
 - JVM manages the location of resources
 - Automatic garbage collection
 - Create resources as needed
 - No need to free
 - JVM manages resources and automatically frees them when they are out of scope

Structure of a simple Java Program

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

- Everything must be in a class
- Java applications (not Applets) must have a main() routine in one of the classes with the signature:
 - **public static void main(String [] args)**
 - public and static defined later
- Compile: `javac HelloWorld.java`
- Run: `java HelloWorld`
- Output: “Hello World”

Scalar Data Types In Java

```
<data type> variable_name;  
int x;
```

- Standard Java Datatypes:
 - byte 1 byte
 - boolean 1 byte
 - char 2 byte Unicode
 - short 2 byte
 - int 4 byte
 - long 8 byte
 - float 4 byte
 - double 8 byte
- Notice the Unicode support and the support for 64 bit (8 byte) numbers

Java is Strongly Typed

- Java is a strongly typed language
- Reduces many common programming errors
- Seems inconvenient or to get in the way for programmers accustomed to programming in C or C++
- Assignments must be made to compatible data types
 - Explicit type casts are required to convert to different types
 - `l = (long) i;`
- Java has clear parameter passing semantics
- In C:
 - no difference between int, char, and boolean
 - Type casts implicit (automatic type promotion) on many types

Expressions

- Java supports many ways to construct expressions (in precedence order):
 - ++,-- Auto increment/decrement
 - +,- Unary plus/minus
 - *,/ Multiplication/division
 - % Modulus
 - +/- Addition/subtraction
- Examples:
 - int x,y;
 - int z;
 - x = 0;
 - x++;
 - y = x + 10;
 - y = y % 5;
 - z = 9 / 5;

Assignment Operators

- Assignment may be simple
 - $x = y$
- Or fancy with the following operators
 - $*=$, $/=$
 - $\%=$
 - $+=$, $-=$
 - $\&=$ (boolean)
 - $|=$ (boolean)
 - $\wedge=$ (boolean)
- Examples
 - `int i = 5;`
 - `i += 10;` `//i = 15`
 - `i %= 12;` `//i = 3`

Conditional Logic

- Conditional logic in Java is performed with the if statement
- Unlike C++ a logic expression does not evaluate to 0 (FALSE) and 1 (TRUE), it evaluates to either true or false
- Keywords **true**, **false** are of data type **boolean**
- Building compound conditional statements
 - && (And), || (Or), ! (Not), <, >, ==, !=, <=, >=, ^ (XOR)

```
int i = 8;
```

```
if (i >= 0) && (i < 10)
```

```
    System.out.println(i + " is less than 10");
```

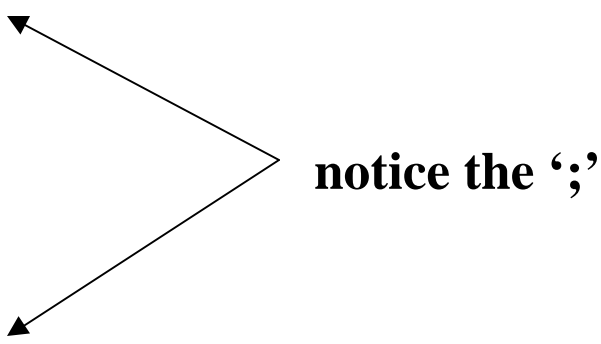
```
else
```

```
    System.out.println(i + " is larger than 10");
```

Code Blocks

- Java, like many other languages allows compound code blocks to be constructed from simple statements
- Just replace a simple statement with:

```
{  
    statement 1;  
    statement 2;  
    .  
    .  
    .  
    statement n;  
}
```



notice the ‘;’

Example

```
int i = 8;  
  
if (i >= 0) && (i < 10)  
{  
    i += 5; // i = i + 5;  
    System.out.println(i + “ is less then 10”);  
}
```

Looping Constructs

- Java supports two looping constructs
 - while
 - do...while
 - for

```
for (int i = 0; i < 10; i++)  
{  
    System.out.println(i);  
}
```

```
int i = 0;  
while(i < 10)  
{  
    System.out.println(i++);    //prints i before  
                                //applying i++  
}
```

```
int i = 0;  
do  
{  
    System.out.println(i++);  
} while(i < 10)
```

I/O

- Java supports a rich set of I/O
 - Network
 - File
 - Screen (Terminal)
 - Screen (Windows, Xterm)
 - Screen Layout - Layout Managers
 - Printer
- For this class we only need to write to the terminal screen
- Use `System.out.println()` or `System.out.print()`
- May use '+' to concatenate

```
int i,j;  
i = 1;  
j = 7;  
System.out.println("i = " + i + " j = " + j);
```

Classes

- A class is an abstract data type (ADT)
- Protects data with functions (methods) that safely operate on the data
- Classes are created with the **new** keyword
- The new keyword returns a **reference** to an **object** that represents an instantiated **instance** of the class
 - Reference is used to operate on the object
- In Java, everything is a class
 - Classes you write
 - Classes supplied by Java specification, other developers, third party companies, Java extensions

```
String s = new String("This is a test")
if (s.startsWith("This") == true)
    System.out.println("Starts with This");
else
    System.out.println("Does not start with This");
```

Methods

- Methods are like functions
- Methods are defined inside of a class definition
- Methods are visible to all other methods defined in the class

```
class foo
{
    int add(int i, int j){
        return i + j;
    }

    int sub(int i , int j){
        return i - j;
    }

    public static void main(Stirng [] args){
        System.out.println(add(1,2));
        System.out.println(sub(5,3));
    }
}
```

Anatomy of a Method

- Visibility identifier
 - public, private, protected
 - We will always use public
- Return type
 - Must specify a data type that the method will return
 - Use the **return** keyword
 - Use **void** if the method will not return any data
- Method name
- Argument list
 - List of parameters
 - Each parameter must be specified by a data type
- Java semantics (IMPORTANT)
 - *Object references passed by reference, all others passed by value*

Class Data

- In Java, we can declare data elements that are global to the class instance (object)
- Invisible/inaccessible to all other classes
- Define at the top of the class
- May be defines as public, private or protected
 - We will use public

```
class foo
{
    int i;
    int j;

    int add(){
        return i + j;
    }

    public static void main(Stirng [] args){
        i = 4; j = 6;
        System.out.println(add());
    }
}
```

Class Data

- Class data is defined to each instance of the class
- Thus each object gets its own copy
- May use the **static** keyword to make a data element common among all class instances

```
class foo
{
    static int i = 0;

    void incrCount(){
        i++;
    }

    int GetCount() {
        return i;
    }
}
```

```
foo f1, f2;

f1 = new foo();
f2 = new foo();

f1.incrCount();
System.out.println(f1.getCount());
f2.incrCount();
System.out.println(f1.getCount());
```

What prints?

What if i was not defined as static?

Constants

- Sometimes it is desirable to create constants to improve code readability
- In C++, done with `const` and `#define`
- In Java, it is somewhat more complicated
 - Define a class data element as
`public final static <datatype> <name> = <value>;`
 - Examples
 - `public final static double PI = 3.14;`
 - `public final static int NumStudents = 60;`
- Constant value can be referenced however it can not be modified

Arrays

- Arrays in Java are classes
- They must be allocated
- They are passed by reference

```
int [] i = new int[10];
```

```
double [] j = new double[50];
```

```
String [] = new String[10];
```

Method Overloading

- In Java, we may create methods with the same name
 - Parameter lists must be different
- Allows for the method to be used based on the called parameters

```
class foo
{
    int add(int i, int j){
        return i + j;
    }

    double add(double i, double j){
        return i + j;
    }

    public static void main(Stirng [] args){
        int i = 4; j = 6;
        double l = 2.1, m = 3.39;
        System.out.println(add(i + j + “,” + l + m));
    }
}
```

Class Constructors

- Each class may have one or more optional constructor(s)
- A constructor is a method (function) that is defined with the same name
- A constructor is automatically called when an object is first created.
- Valuable for initialization of critical member data

```
class foo
{
    String m_s;

    int foo(){
        m_s = "hello world";
    }

    ...

}
```

Packages

- Package is a loose affiliation of classes
- Packages are a primary mechanism for adding additional functionality into Java
- Packages like libraries
- You may create your own packages with the **package** keyword
- We will focus on using some thread creation and synchronization packages in this class
- Packages are located by the CLASSPATH environment variable
- In order to use a package you use the **import** keyword.

Some Common Java Packages

- java.applet
- java.awt
- java.io
- java.lang
- java.net
- java.util
- java.math
- java.security
- java.beans

Good idea to import the classes that you need

```
import java.io.*;  
DataInputStream dis = new DataInputStream()
```

or

```
java.io.DataInputStream dis = new  
    java.io.DataInputStream()
```

The String Class

- Used to create and manipulate strings in Java
- Better than a null terminated sequence of characters
- Automatically sized (automatic storage management)
- Rich set of functions
 - Concatination
 - Lowercase/Uppercase data conversion
 - Substrings
 - Strip leading and trailing characters
 - Length
 - Data conversion to/from an array of character representation

The Vector Class

- The vector class is very useful as a dynamic array
- The vector class can hold any java object

```
import java.util.Vector;
```

```
Vector v = new Vector();
```

```
v.removeAllElements();
```

```
v.addElement(new String("1"));
```

```
v.addElement(new String("2"));
```

```
v.addElement(new String("3"));
```

```
v.addElement(new String("4"));
```

```
for (int i = 0; i < v.size(); i++)
```

```
    System.out.println("Data = " + v.elementAt(i));
```

Data Conversion Functions

- Java contains a robust set of classes to convert from one data type to another
- See reference, however there is a data encapsulation class for each scalar type
 - Long for long
 - Integer for int
 - Float for float
 - Double for double
 - Boolean for boolean
- Typical functionality includes
 - Convert to/from bases
 - Convert to/from a string
 - Equality checking

Simulating Modules in Java

- A module is a collection of related functions
- Although not “pure” object-oriented - a class can be used to simulate or package a module:
 - Create class with main routine
 - Place supporting functions in the module inside the class as **public** methods
 - Define any global module data as class data elements

Concurrent Programs

- Concurrent programs contain more than one thread of execution
- Supported by the Java **Thread** class and the **Runnable** interface
- To make a class behave as a Thread
 - Implement the **Runnable** interface
 - An interface is used to specify methods that your class must support
 - The **Runnable** interface requires that any class implementing this interface have a method named:

run()

- To create a thread you must create an instance of the thread class passing an instance of your class to the constructor
- See following example

Thread Example

```
class T1 implements Runnable {
    Parent m_p;

    T1(Parent p)
    {
        m_p = p;
    }

    public void run ()
    {
        int before, after;
        Long sval;
        double sLen;

        sLen = Math.random() * (double)500.0;
        Double rLen = new Double(sLen);
        while (true)
        {
            ...
        }
    }
}
```

Thread Example

Thread main body:

```
while (true)
{
    before = m_p.GetValue();
    try
    {
        Thread.sleep(rLen.longValue());
    } catch (InterruptedException e)
    {
        System.out.println(e.toString());
    }
    after = m_p.GetValue();
    m_p.IncrValue();
    System.out.print(
        "(" + before + ", " + after + ") ");
    System.out.flush();
}
}
```

Thread Example

```
class Parent
{
    int m_val = 0;

    public int GetValue()
        { return m_val; }

    public void IncrValue()
        { m_val++; }

    public void RunSimulation()
    {
        Thread pa = new Thread(new T1(this));
        Thread pb = new Thread(new T1(this));
        Thread pc = new Thread(new T1(this));
        pa.start(); pb.start(); pc.start();
        try{
            Thread.sleep(9000);
        }catch(InterruptedException e) {
            System.out.println(e.toString());
        }
        pa.stop(); pb.stop(); pc.stop();
    }
}
```

Thread Example

```
public static void main(String[] args)
{
    Parent p = new Parent();
    p.RunSimulation();
}
}
```

- Must create an instance of the Parent class
 - Due to the main routine being static
- Once Parent object created we can call the run simulation method

Thread Example Results

(101,103) (99,104) (103,105)
(111,111) (110,112) (109,113)
(114,119) (119,120) (118,121)
(126,127) (124,128) (128,129)
(134,135) (133,136) (136,137)
(142,143) (140,144) (143,145)
(149,151) (151,152) (150,153)
(154,159) (158,160) (160,161)
(166,167) (165,168) (167,169)
(173,175) (174,176) (177,177)
(182,183) (183,184) (181,185)
(189,191) (192,192) (191,193)
(197,199) (196,200) (199,201)

Note that both numbers in the pairs above should be the same. They differ due to race conditions

For More Information

- There are many books and online references for Java (check out: **Java** by Wrox Press)
- To use Java to its maximum potential you should first study OO theory
 - Inheritance
 - Polymorphism
 - Interfaces
 - Abstract Base Classes
 - Protection and Invariants
- Java now has component and distributed component support
 - Java Beans, Enterprise Java Beans
- Java now has CORBA interoperability
 - IDL to generate stub/proxy interface code
- Java has solid relational database interfaces
 - JDBC, JDBC/ODBC bridge

Java Futures

- Excellent commercial IDE's continue to evolve
 - Jbuilder
 - Visual J++
 - IBM VisualAge for Java
- COTS
 - Applets, beans, components
- Embedded applications, custom Java hardware
 - See “PicoJava: A Direct Execution Engine for Java Bytecode”, October 1998 IEEE Computer
- More value added services
 - Imaging
 - Compression
 - Security/Cryptography
 - Realtime video

Java Challenges

- Key challenge - Keeping the language standard and portable
 - Must force vendors to add features through defined mechanisms for extension
- Already issues between Sun and Microsoft around the Java language
 - Legal battle for over a year
- Sun submitting Java language to standards committee