CS 390 Unix Programming Environment

Topics to be covered:

- Introduction to programming in Java
Introduction to Java

- Invented in early 90’s by a team at Sun Microsystems, headed by James Gosling
- Due to growth of the Internet, there was a need to develop applications that are platform independent.
- Internet connects a wide variety of machines running under different platforms.
Java Features

- Java is an Object-Oriented language
- “Write Once Run Anywhere”- WORA
- Portability is because of Java virtual machine technology:
  - Byte Code
    - Byte code is neutral and does not pertain to any particular computer
- JIT (Just In Time) compilers
  - Interpreted
  - Translates the byte code into native instruction set understandable by the computer
Java Features

- The Magic of Byte Code
  - All primitive instructions in Java are called *bytecodes*
  - Byte codes bring the compiled instructions to lowest level possible without making them machine dependent
  - These bytecodes make Java a partially compiled language
  - Creating the bytecodes is about 80% of compilation code and 20% is performed during the run time

CS390 - Summer 2000
Java Features

Java at Run-time

- As soon as the bytecodes reach the target machine, they’re run through a Java verifier.
  - Bytecode verifier tests the bytecode received for forged pointers, access violations etc...
  - Class loader loads the classes and gives to the interpreter

Interpreter uses JIT techniques

- Translates a class into native code prior to execution

Finally, there are java compilers that translate Java directly to (.EXE) binaries.
Virtual Machine

- A byte code interpreter intended to execute the byte code produced by the Java compiler is called a Java virtual machine.

- Once a Java virtual machine has been implemented for a particular computer, that computer will run any compiled Java program.

- Virtual machines are platform-specific. For example a Java virtual machine for Sun O/S might not run under Windows NT.
Java Environment

Java Application

Java VM (interpreter)

JIT Compiler

byte code

native code

Dynamic compilation

OS/2  Win32  Linux  AIX  Network Station  AS/400  OS/390

Intel x86  PowerPC  S/390

CS390 - Summer 2000
Java Features

- Similar to C++, but cleaner
- No pointers, typedef, preprocessor, structs, multiple inheritance etc...
- Automatic garbage collection
- Java programs can be loaded dynamically via a network
- Java programs can work on multiple tasks simultaneously

CS390 - Summer 2000
Java Language

- Java is a very powerful language.
- We will focus on a minimal set of features of Java that will help us develop an application.
- Input and Output will be (character) terminal based.
  - Java has AWT and Swing for fancy UI, but will not be covered in this tutorial
Java Virtual Machine (JVM)

- All Java programs run within a Java Virtual Machine (JVM).

Features

- Security
- Portability
- Dynamic Resource Management
- Resource location transparency
- Automatic Garbage Collection
  - Create resources when needed
  - No need to free them, it's automatically freed when they become out of scope
Structure of a simple Java Program

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

- The name of the file must be <classname>.java
  - E.g. the above program’s file name must be HelloWorld.java (Note that Java is a case sensitive lang.)
- Every thing must be in a class
- Java come in various flavors
  - Applications, applets etc.
- All Java Applications must have a main routine with the signature:
  - public static void main(String args[])
Structure contd...

- Compile as follows: `javac HelloWorld.java`
- Run: `java HelloWorld`
- Output: `HelloWorld!`
Standards

- A Java program may contain one or more classes.
- However, there must be only one class having one `main` method.
- The class file that will be fed to the java interpreter will be the class having the `main` method.
Scalar Data Types in Java

<data type> variable_name;

E.g. int x;

Standard Java DataTypes

- byte 1 Byte
- boolean 1 Byte
- char 2 Byte Unicode
- short 1 Byte
- int 4 Bytes
- long 8 Bytes
- float 4 Bytes
- double 8 Bytes

Notice the support for unicode 64 bit (8 bytes) numbers

CS390 - Summer 2000
Java is strongly typed

- Java is a strongly typed language
- Reduces many common programming errors
- Assignments must be made only to compatible data types
  \[ L = (\text{long}) \ i; \]  //L is of type long and I is of type int
- Java has a very clear parameter passing semantics which C lacks
Expressions in Java

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;`
Assignments in Java

Assignments may be simple

\[ x = y; \]

Or may be fancy, as you have seen in C++;

\[ *= , +=, %=, &=, |=, ^= \]

Examples:

```java
int i = 5;

i += 10;    // i = 15;
i %= 12;    // i = 3;
```
Comments in Java

Java Supports 2 commenting styles

```
// : Line comment
/* : Block comment
```
Conditional Logic

- Conditional logic in Java is performed with the `if` statement.
- Unlike C++, a logical expression does not evaluate to 0 (false) or 1 (true) it evaluates explicitly to `true` or `false`.
- Keywords, `true` and `false` are of data type `boolean`.
- We can build compound conditional statements using `&&`, `||`, `<`, `>`, `!=`, `<=`, `>=`, `==`, `^` (XOR).

```java
int i = 8;
if ((i >= 0) && (i < 10))
    System.out.println("The number "+i+" is less than 10");
else
    System.out.println("The number "+i+" is greater than 10");
```
Code Blocks in Java

Java, like other programming languages allows compound code blocks using simple statements:

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

Example:
```
int x = 8;
If( (x > 0) && (x <= 10) ) {  
x += 5;
    System.out.println(" The value of x is "+x);
}
```

CS390 - Summer 2000
Looping Constructs

Java supports following looping constructs:

- for
- do while
- while

These constructs are similar to the ones in C++

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

Exercise: Try do while
Java I/O

- Java supports rich set of I/O:
  - Network
  - File
  - Printer
  - Windows Screen, Screen layout, Terminal

- In this class we will be focusing on file and terminal
- Use `System.out.print` and `System.out.println` for printing to terminal
  ```java
  System.out.println("The value " + x + " is not valid");
  ```
- For inputs to the program, we make use of command line arguments as follows:
  ```java
  java MyProgram arg1 arg2
  ```

at the command prompt
Java I/O contd...

- The arguments are stored in `args[]` array.
- Note that arguments are considered as `String` objects.
- The index for the `args[]` starts from 0.
- In the program, an argument is read as follows:
  
  ```java
  String Name = args[0];
  ```

- In case, you are reading an integer from the command line, we will have to convert it from a `String` to a primitive `int` data type using `parseInt()` method of `Integer` class as follows:
  
  ```java
  int x1 = Integer.parseInt(args[1]);
  ```

- Let's now discuss what is a Class.

  CS390 - Summer 2000
Classes

- A class in java is an ADT
- Protects data with methods (operations)
- Classes are created using `new` keyword
- The `new` keyword returns a reference to an object that represents the instantiated instance of the class
  - This reference is used to operate on the object
- In Java everything is a class
  - Classes you write
  - Classes supplied in Java Specs, third party, developers...

```java
String S = new String("This is CS390");
if((S.startsWith("This") == true)
    System.out.println("Starts with true");
else
    System.out.println("Does not start with true");
```
Methods

Methods are like functions
Methods are defined inside of a class definition
Methods are visible to all other methods of that class

```java
class foo {
    int add(int i, int j) {
        return i + j;
    }
    int sub(int i, int j) {
        return i - j;
    }
    public static void main(String args[]) {
        System.out.println(add(5, 3));
        System.out.println(sub(8, 4));
    }
}
```

CS390 - Summer 2000
Anatomy of a method

- Visibility identifier
  - public, private, protected

- Return type
  - Must specify what data type the method will return
  - Use the return keyword
  - Use void if nothing is returned

- Method Name

- Argument list
  - List of parameters
  - Each parameter must be specified by a data type

- Java Semantics (IMP)
  - All Object references are passed by reference, all others are passed by value

public void CheckName(int x, String Name);
Class Data

In Java, we can declare data elements that are visible to all the class methods
Invisible / Inaccessible to other classes
Define at the top of the class
May be defined as public, private, or protected
  • We will be using public

class foo {
    int i;
    int j;

    int add() {
        return i + j;
    }

    int sub() {
        return i - j;
    }

    public static void main(String args[]) {
        i = 4; j = 6;
        System.out.println(add(i, j));
    }
}
Class Data

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

```java
class foo {
    static int x = 0;
    void incrCount() {
        x++;
    }
    int getCount() {
        return x;
    }
}
```

```java
foo f1, f2;
f1 = new foo();
f2 = new foo();
f1.incrCount();
System.out.println(f1.getCount());
f2.incrCount();
System.out.println(f2.getCount());
```
Constants

- Sometimes it is desirable to create constants to improve code readability.
- In C++ it is done using `#define`.
- In Java, it is somewhat complicated.
  - Define constants as:
    ```
    public final static double PI = 3.14;
    ```
- These constants can be referenced, however they cannot be modified.
Arrays

- Arrays in java are classes
- They must be allocated
- They are passed as reference, just like in C++

```java
int[] i = new int[10];
double[] j = new double[15];
String[] s1 = new String[10];
```
Method Overloading

- In Java, we can create methods having same names.
- However, they must have different parameter list.

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

CS390 - Summer 2000
Class Constructors

- Each class may have one or more optional constructors
- A constructor is a method that has the same name as that of the class
- A constructor is automatically called when an object is created
- Valuable for initialization of critical member data

```java
class foo {
    String m_s;
    foo() {
        m_s = "Hello!";
    }
}
```
Using Files

- The access files, we make use of FileInputStream class for reading from a file and FileOutputStream class to write to a file.
- We create a FileOutputStream as follows:
  FileInputStream outFile = new FileOutputStream(“data1.dat”);
- Similarly, we can create a FileInputStream as follows:
  FileInputStream inFile = new FileInputStream(“input1.dat”);
- In order to print to a file we have to make use of PrintStream class:
  PrintStream output = new PrintStream(outFile);
  output.print(“ Hello !”);
- Similarly, to read from a file we make use of StreamTokenizer class as follows:
  StreamTokenizer tokens = new StreamTokenizer(inFile);
Packages

- Package is a loose affiliation of classes
- Packages are the primary mechanism of adding additional functionality into java.
- Packages are like libraries
- One can create a package using the `package` keyword
- In order to use a package use `import` keyword
Some common Java Packages

```java
import java.io.*;
import java.awt.*;
import java.lang.*;
import java.util.*;
```

Good idea to import only libraries that you need:

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

OR

```java
java.io.DataInputStream dis = new java.io.DataInputStream();
```
The String Class

- Used to create and manipulate strings in Java
- This is better than a null terminated sequence of characters.
- Automatic sized
- Has rich set of functions:
  - Concatenation
  - Lowercase / uppercase conversion
  - Length of string
  - Substrings

- Please refer to the O’Reily Java reference

CS390 - Summer 2000
The Vector Class

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

```java
import java.util.*;
Vector v = new Vector();
v.removeAllElements();
v.addElement(new String("1"));
v.addElement(new String("2"));
v.addElement(new String("3"));
for(int i = 0; i < v.size(); i++)
    System.out.println(v.elementAt(i));
```

CS390 - Summer 2000
Data Conversion

- Java contains robust set of classes that help in converting from one data type to another
- See the reference book for more details
- Typical functionality includes:
  - Convert to/from bases
  - Convert to/from String
  - Equality checking
Concurrent Programs

- Concurrent programs contain one or more thread of execution
- Supported by the Java **Thread** class and **Runnable** interface.
- To make a class behave like a thread:
  - Implement a **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:
    ```java
    run()
    ```

- See the following example:

  CS390 - Summer 2000
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) {
            ...
        }
    }
}

CS390 - Summer 2000
Thread Example

Thread main body:
while (true) {
    before = m_p.GetValue();
    try { // defining exceptions
        Thread.sleep(rLen.longValue());
    } catch (InterruptedException e) {
        System.out.println(e.toString());
    }
    after = m_p.GetValue();
    m_p.IncrValue();
    System.out.println("("+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));
        try {
            Thread.sleep(9000);
        } catch (InterruptedException e) {
            System.out.println(e.toString());
        }
        pa.stop; pb.stop; pc.stop;
    }
}

CS390 - Summer 2000
Thread Example

```java
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 the parent object is created we can call the `RunSimulation()` method
- Must try this example and understand clearly the working of this example
- As an assignment, run this example and the other sample programs and bring the output in the next class. You might be asked to explain the working of the program.

**Extra Credit: 2 points**

CS390 - Summer 2000