CS 390 Unix Programming Environment

Topics to be covered:

- Introduction to Object-Oriented programming in Java
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");
```
Scope of Classes

- A class declaration is as follows
  \[
  \langle \text{scope} \rangle \ \langle \text{type} \rangle \ \text{class} \ \langle \text{class\_name} \rangle \{ \\
  \}
  \]

Scope: public, protected
Type: Abstract, Interface, Instance
- We will discuss scope when discussing class data and methods
- A class cannot be of private type
Form of a class

- One can define a class as follows:

```java
class classname {
    type instance-vari;
    type instance-vari2;

    type methodname(parameter-list) {
    }
}
```

- Objects are created using the `new` keyword

```java
Class1 cs1 = new Class1();
```
Example of a class

class Box{
    double width;
    double height;
    double depth;
    double calcVolume();
}

So an Object of type Box is created as follows:
Box B1 = new Box();
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));
    }
}
```
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

```java
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
public 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());
```
Scopes

- A scope of class data & methods can be either public, private or protected.

- Using the private variables and public methods, one can access the private variables indirectly from:
  - Anywhere in the class
  - Compilation unit
  - Package
  - Or some other package

- Instance variables and instance methods in the protected part of a class definition are accessible from:
  - Same class
  - Compilation unit
  - Package
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:
    ```java
    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

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

- Each class may have one or more optional constructors
- Java provides a do-nothing default constructor if a constructor is not defined
- 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!";
    }
}
```
Inheritance Basics

- A class that is inherited is called as *superclass*
- The class that does the inheriting is called a *subclass*
- One can inherit the properties of a superclass by using the keyword **extends**
- For example,
  ```java
  class B extends A {
      ...
  }
  ```
- A base class can call a constructor method defined by its superclass by using the keyword **super**.
- For example,
  ```java
  super(w,h,d); // call superclass constructor
  ```
Inheritance contd...

- A subclass can override methods of a superclass
- To prevent overriding, a superclass must define the method of type **final**
- For example,
  ```java
  class A{
    final void meth() {
      ...
    }
  }
  ```
- Java does not support multiple inheritance
Abstract Class

- One can define abstract classes in Java using the `abstract` keyword.
- For example,

```java
abstract class A{
    abstract void callme();
    void callmetoo() {
        System.out.println("This is a concrete method");
    }
}
```
Abstract Class contd...

class B extends A {
    void callme() {
        System.out.println("B’s implementation of callme");
    }
}

A base class need not implement all the methods of a super class
Interfaces

- One can also create pure abstract classes using **interfaces**.

```java
interface callBack {
    void callback(int param);
}
```

- To implement the interface, include the **implements** clause
Interfaces contd...

class client implements Callback {
    public void callback(int p) {
        ...
    }
}

Note that a subclass must implement all the methods of an interface class
Inheritance – Class Hierarchy

- As mentioned earlier everything in Java is a class
- A class hierarchy exits in java
- The “Object” Class appears top-most in this hierarchy
- All Java classes inherit methods and data members from its super class
- All other Java classes have one direct superclass except for the Object class
Handling Exceptions

- A Java exception is an object describing an error condition.
- When an exception is *thrown* in a method it is *caught* and processed.
- Java exception handling is done using these keywords:
  - `try{ }`: block of code to monitor for errors
  - `catch(excep exOb) {}`: Exception handler for Exception
  - `finally {}`: block of code to be executed before try block ends
class MultiCatch {
    public static void main(String args[]) {
        try {
            int a = args.length;
            System.out.println("a = " + 1);
            int b = 42 / a;
            int c[] = { 1 };
            c[42] = 99;
        } catch (ArithmeticException e) {
            contd...
        }
    }
}
Exception example contd...

```java
System.out.println(" Divide by 0: "+e);
} catch (ArrayIndexOutOfBoundsException e) {
    System.out.println("Array index oob: " +e);
}
finally {
    System.out.println(" Handled all exceptions...");
}

Java Supplies rich amounts of in-built exceptions
Check the reference book for more details
There are also user-defined exceptions, i.e. users can create their own exceptions using
class MyException extends Exception {...}
```
Exceptions – Another way

n Java provides an alternate way to define exceptions

n For example, if you are using file I/O in your program, Java forces you to define the IOException, which can be defined as follows:

```java
public class Demonstrate {
    public static void main(String args[]) throws IOException {
        ...
    }
}
```
Using Files

- To 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:

  ```java
  FileOutputStream outFile = new
  FileOutputStream("data1.dat");
  ```

- Similarly, we can create a `FileInputStream` as follows:

  ```java
  FileInputStream inFile = new
  FileInputStream("input1.dat");
  ```

- In order to print to a file we have to make use of `PrintStream` class:

  ```java
  PrintStream output = new PrintStream(outFile);
  Output.print(" Hello !");
  ```

- Similarly, to read from a file we make use of `StreamTokenizer` class as follows:

  ```java
  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
Short Package example

package MyPack;

class Balance {
    String name;
    double bal;
    ...
}

Save this as Balance.java in MyPack directory
Using this Package

```java
import MyPack.*;

class TestBalance {
    Balance test = new Balance();
    ...
}
```
Some common Java Packages

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

Good idea to import only libraries 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
- 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
Some important String Methods

- `equals` - Compares two strings, returns true if equal, false otherwise
  - Usage: `str1.equals(str2)`

- `length` - returns length of a string
  - Usage: `str1.length()`

- `concat` - concatenates two strings and returns the result
  - Usage: `str1.concat()`
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));
```
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
r
  run()
  ```

- See the following 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 { // 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 38
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.
Assignment No. 4

- Run the above Thread Example
- Run all the programs that are listed on the reference link
- Explain them and prepare a report having following things:
  - Prepare a header with description of the program
  - Create sample test cases and run the program using them
  - Print the test cases and the run for each program