V22.0490.001
Special Topics: Programming Languages

B. Mishra

New York University.

Lecture # 24

—Slide 1—

Classes

- Similar to C++ classes
- All classes are derived from the class `Object`
- Instance variables and methods can be `public`, `protected`, or `private` (same definitions as C++).
- Anything not declared as `public`, `protected`, or `private` is visible throughout its `package`.
  This is the only way to get "friendly" behavior
- *Single Inheritance*
  Each class (except `Object`) has exactly one superclass.
- Initialized to `null`. Takes on value only through the = operator or the `new` operator.

—Slide 2—

Static and Final Methods and Variables

- Static variables and methods are attached to a specific `class`
  –Instead of, to a particular instance of a `class`
  –They are accessed through the `class` name (instead of through a variable)
- Static methods may not reference instance variables or methods
- Final variables must be initialized and may not be changed.
  They are constants
- Final methods may not be overridden.

—Slide 3—

Abstract Classes and Method

- An abstract class is one that cannot be instantiated—only subclassed.
- Abstract class provides prototypes for methods that it does not implement.
- An abstract class definition is preceded by the word `abstract`.
- An abstract method (also preceded by the word `abstract`) is one which is not implemented.

—Slide 4—

Java Example

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

- `HelloWorld` is defined to be `class`
- No instance variable
- A single `public` method called `main`. `main` is static—It is attached to the class itself and not an instance of the class
- The method `main` contains in its body a single method invocation to display the string `"Hello World!"` on the standard output

—Slide 5—

Classes in Java: Example

- A class defines the instance variables and methods of an object. It is a template that defines how an object will look and behave when it is instantiated.

```
class Point{
   public float x;
   public float y;

   Point(){
      x = 0.0;
      y = 0.0;
   }
}
```

- *Instantiation*

```
Point myPoint;
myPoint = new Point();
```

- *Manipulation*

```
myPoint.x = 10.0;
myPoint.y = 25.7;
```

—Slide 6—

Classes in Java: Example (Contd.)

- *Constructor* Performs initialization when you instantiate objects from a class

```
public final class Integer extends Number{
   private int value;
   public Integer(int value){this.value = value;}
}

Integer myIntegerObject = new Integer(123);
```

- *Finalizer* Performs necessary “tear-down” (or “wills-and-testament”) before the garbage collector is about to free the object

```
   protected finalize(){
      try{
         close();
      } catch (Exception e){ }
   }
```

—Slide 7—

Strings

- Provided as a Class in the `java.lang` package.
- Not just a string of `char`s.
- Provides `+` operator (concatenation).
- String length is fixed by constructor.
- Use `StringBuffer` class for variable length strings.

—Slide 8—

Interfaces

- An interface specifies a group of method prototypes and field variables
- All field variables are implicitly `static` and `final` and must be initialized with a constant expression
- An interface, I, may extend other interfaces. Any class that implements I must implement all the interfaces that I extends
- A class that implements the interface must instantiate each method in the interface
- A variable of type interface v can be instantiated with a reference to any class that implements v
- Interface involves dynamic method binding—There is a small performance penalty to using them
- *Combining Interfaces* Interface can incorporate one or more other interfaces (using `extend`)—This gives multiple inheritance over the interfaces.

—Slide 9—

Interfaces: Example

```
public interface Storing {
   void freezeDry(Stream S);
   void reconstitute(Stream S);
}
public interface Painting{
   ...
}

public class Image implements Storing, Painting {
   ...
   void freezeDry(Stream S){//JPEG compression of image...}
   void reconstitute(Stream S){//JPEG decompression of image...}
}

interface DoesItAll extends Storing, Painting {
   void doesSomethingElse();
}
```

—Slide 10—

Arrays

- Each class type has an array type created automatically
- Array type declared by adding `[]` to variable
- Initialization through the `new` operator or `=`
- All arrays are single-dimensional. Must use arrays of arrays instead of multi-dimensional arrays
- An array is an object with a number of variables. Instead of having names, these variables are referenced by non-negative integers (their *indices*)
- The array length is not part of its type—Thus, over its lifetime, a given array variable may refer to arrays of different lengths
- Every array has a `.length` field, which is a final variable. Once an array object is allocated, its length never changes.
- Array bounds are checked at run-time; `ArrayIndexOutOfBoundsException` is thrown if an attempt to reference an index out of the range `[0..length-1]` is made

—Slide 11—

Storage Class

- Determines lifetime of a variable.
- **Local Variables:** Declared and allocated within a block.
  –Discarded at end of block
  –Method parameters are considered local.
- **Static Variables:** Local to a class.
  –Allocated when class is loaded and
  –Discarded when class is unloaded.
- **Dynamic Objects:** Instances of classes and arrays.
  –Allocated by `new` expression
  –May be referenced by more than one variable
  –Garbage collector handles reclamation of storage used by dynamic objects.

—Slide 12—

Structure

- **Package**
  –Made up of compilation units
- **Compilation Unit**
  –File made up of classes and interfaces with at most one public class or interface

—Slide 13—

Exceptions

- `try` block followed by one or more `catch` blocks.
- If an exception occurs in a `try` block, the following `catch` blocks are examined
- The first `catch` block whose argument type matches the exception is executed
- A `finally` clause may be attached after the `catch` blocks—In this case, the code in the clause gets executed after any `catch` block is executed.

—Slide 14—

Java Class Libraries

- `Language Foundation Classes`
  –Wrappers for primitive types and fundamental classes. Also Math routines
- `I/O Class Library`
  –File and Stream input and output.
- `Another Window Toolkit Class Library`
  –Everything you need to build a GUI.
- `Utility Class Library`
  –Implements a variety of encoder and decoder techniques, data and time, hash table, vector, and stack.
- `Network Interface Class Library`
  –Extends the functionality of the I/O class library with socket interfaces and Telnet interfaces

—Last Slide—

What Java Lacks

- *Header files, typedefs, #define, and preprocessor* –Makes Java more context free
- *Structures and Unions* –Just use classes.
- *Functions* –Force programmers to stick to objects
- *Multiple Inheritance* –Interfaces address some of these capabilities
- *Goto statement*
- *Operator Overloading*
- *Automatic Coercions* –Must explicitly cast if a loss of precision may occur.
- *Pointers* –Cause of buggy code; major security hole.
- *Templates*

[End of Lecture #24]