Practice of Programming using Java

Lecture 2 & 3
June 16, 2006
6-8pm LT

About JAVA

- An Object Oriented Programming Language
- An Interpreted Language
- Portable Programs
- Internet Enabled
- Provides a large number of Library Packages for software Development

The *Class* Construct

- Put together data/implementation and interface, and enforce encapsulation as per user’s design
- A class provides the definition of an object
  - Object’s state
  - Object’s behavior in terms of public methods
- Many instances (objects) can be created from a given class. All the instances behave as specified by their class

Initialization and Destruction

- A class also defines a special initialization method for its objects so that whenever objects are created, they are initialized automatically
- Destruction is automatic through garbage collector. However:
  - A class can defines a destroyer method which is called when an object turns into garbage

Compiling and Executing JAVA programs

- Myclass.java   contains a class called Myclass.
- Compile this file to a Myclass.class file using javac, a Java compiler that compiles Java source code to intermediate Byte code.
- If Myclass supports a special method called main, you can ‘execute’ Myclass.class with Java interpreter.

An Example Java Class

```
class Pair {   // this is a comment
private int x, y;
public Pair (int i, int j) { x=i; y=j;} // initialize
public void print ( ) {
        System.out.println (x+” “+y);
   }
}
```

Use class Pair

```
class TestPair {
  public static void main (String arguments[]) {
    Pair p1 = new Pair (10,20);
    p1.print ();
  }
}
```

Javac Pair.java	*Produces Pair.class*
Javac TestPair.java	*Produces TestPair.class*
Java TestPair	*Executes main in TestPair.class*

Hallo.java

```
Class Hallo {
   public static void main (String args[]) {
        System.out.println (“Hallo “ + args
   [0]);
   }
}
```

javac Hallo.java --> compiles to Hallo.class

Constructor

- *Constructors have input arguments but no return types*

```
class Stack {
   int state[], max, top;
   Stack (int m)  { max = m;
                    top = 0; }
}
```

If you don't specify one, the system constructs a default constructor.

Finalizer

- Is not a destructor as in C++
- Finalizer is called just before garbage collector takes over the object
- Use for file close operations, resource closing ensuring safe reclaiming of resources
- *protected void finalize () {*

  …… ; super.finalize;   // a good practice

  }

Primitive Types

boolean	*true or false*
byte, short, int, long	*8..64 bit integers*
float, double	*32..64 bit real*
char	*16 bit unicode*

Arrays

- *They are First Class Objects*
- *Example:*

```
int [] state;
for (I=0; I< state.length; I++)
        state [I] = 100 ;
```

Exercise: convert the above loop into an equivalent while loop

Relational Operators

> <	greater than?, less than?
>= <=	gteq? Leq?
==	equal?
!=	not equal?
instanceof	*Type comparision*

Bitwise and Boolean Operators

- ~ Bitwise C*omplement*
- & Bitwise AND
- ^ Bitwise EXOR
- | Bitwise OR
- && Boolean And
- || Boolean OR

Shift Operators

- << left
  - x << y shift x left by y bits
  - Fills 0s on right
- >> right
  - x >> y shift x right by y bits
  - Fills higest bit on left
- >>> *0 fill right shift*
  - *X >>> y shifts x right by y bits*
  - *Fills 0s on left*

Other Operators

- *?:*
  - *X ?: y : z returns y if x is true else returns z*
- *(type)*
  - *(t) x casts x to type t*
  - *Tx x*
  - *y = (Ty) x*
- *Instanceof*
  - *x instanceof y returns true if x is instance of class y else it returns false*

Arithmetic and Assignment Operators

- + - / * %
- =
- ++ --
- += -= *= /= %=
  - *operations + assignment*
- &= ^= |= <<= >>= >>>=
  - *bitwise operations + assignment*

Control Flow

- if (…) …… else ……. ;
- switch (….) { case … };
- while (…) ….;
- do …. while (…);
- for (… ; …; …;) ….. ;
- *label* : …

  break *label* or continue *label*
- return (…) ; return ;

Parameter Passing

All parameters are Call By Value

- Primitive types

  a parameter of primitive type is a copy of its value

- Reference types

  a parameter of this type is a copy of the reference and not the object to which it points

What are static members ?

They are 'Per Class' members, shared by all objects of that class

- static data members (fields) : *class variables*
- static methods *(*class methods*)* : *can access only class variables and static methods*

main is always specified as static: use main as a class tester

Static initializers

- T*o initialize static fields such as arrays*
- *complex static initialization can be done through this function*

Static Initializers

```
class Student {
    private static int  n;

    static  {
                    n = 0;
    }
    public static int count() { return n;}
…
}
```

Standard JAVA Utility Classes

- These are many useful classes provided by the java environment in java.util package
- For example:
  - BitSet: Is a collection of bits
  - Vector: Is a dynamically sized array of Objects
  - Stack: LIFO vector

Class BitSet

- Represents a bit vector (of true/false bits) that can grow dynamically

- An Example:

```
BitSet b = new BitSet(2);  or
BitSet b2 = new BitSet();
b2.set(1); b.clear(0);
```

Some Methods on BitSet

void set (int p)

sets bit at position p to true

void clear (int p)

sets bit at position p to false

boolean get (int p)

returns bit a position p

void and (BitSet other)

Logical ANDs *this* bitset with *other*

Some Methods on BitSet

void or (BitSet other)

void xor (BitSet other)

int size ()

returns size of the bitset

boolean equals (Object other)

returns true if bits in other are same as those in this

Class Vector

- Represents a resizable array of Object References
- Arrays in JAVA are of fixed size whereas vectors are resizable. But vectors can hold Object references and not primitive data types
- Use vectors when you do not know in advance the size of your array

An Example

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

for (i=0, I<n, I++) {
   s=new Student (i);
   v.addElement (s);
}
```

Some Methods on Vector

final synchronized void addElement (Object element)

adds an element into the vector

final synchronized boolean removeElement(Object element)

removes the element from the vector, if the object is not found, value false is returned

final synchronized void insertElementAt (Object element, int position)

inserts an element at given position, elements after position are moved up

Some Methods on Vector

final synchronized void removeElementAt (int position)

removes element at given position, elements after the position are moved down

final synchronized void setElementAt (Object element, int position)

sets *element at given position*

final synchronized void removeAllElements ()

clears all elements and makes the vector empty

Some Methods on Vector

final int size ()

returns the size of the vector

final boolean isEmpty()

returns true if the vector is empty

Some Methods on Vector

final synchronized Object elementAt (int position)

returns element at given position

final boolean contains (Object element)

indicates whether the element is contained in this vector

final int indexOf (Object element)

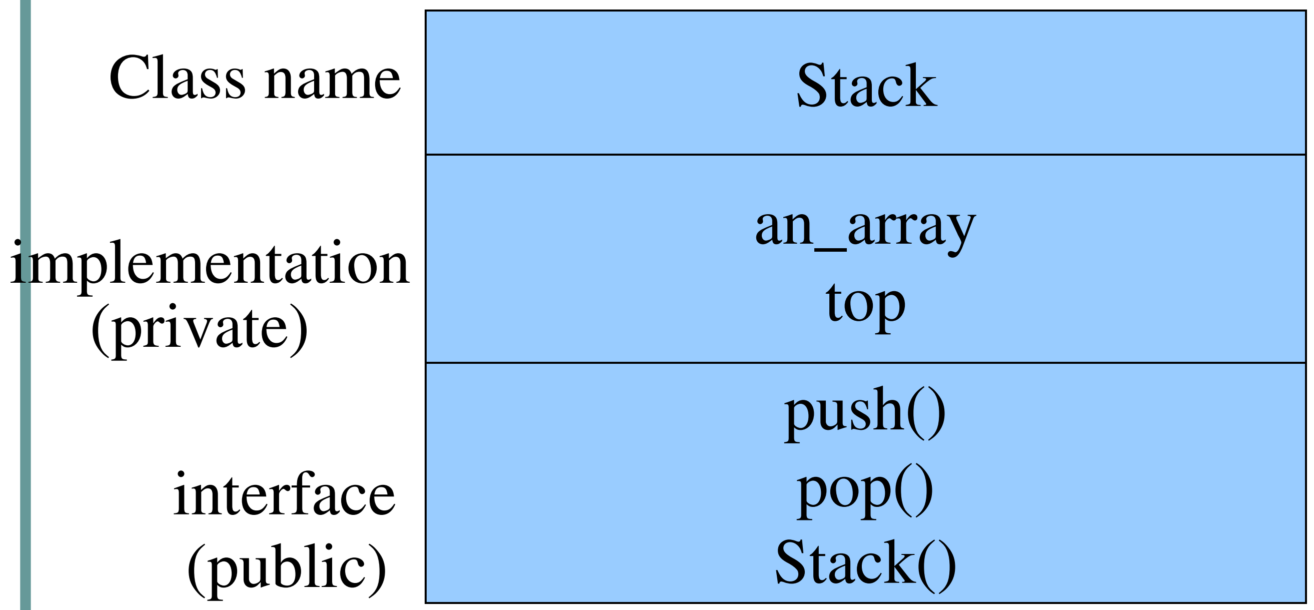
returns the position of element,-1 if not found

Developing Class Stack

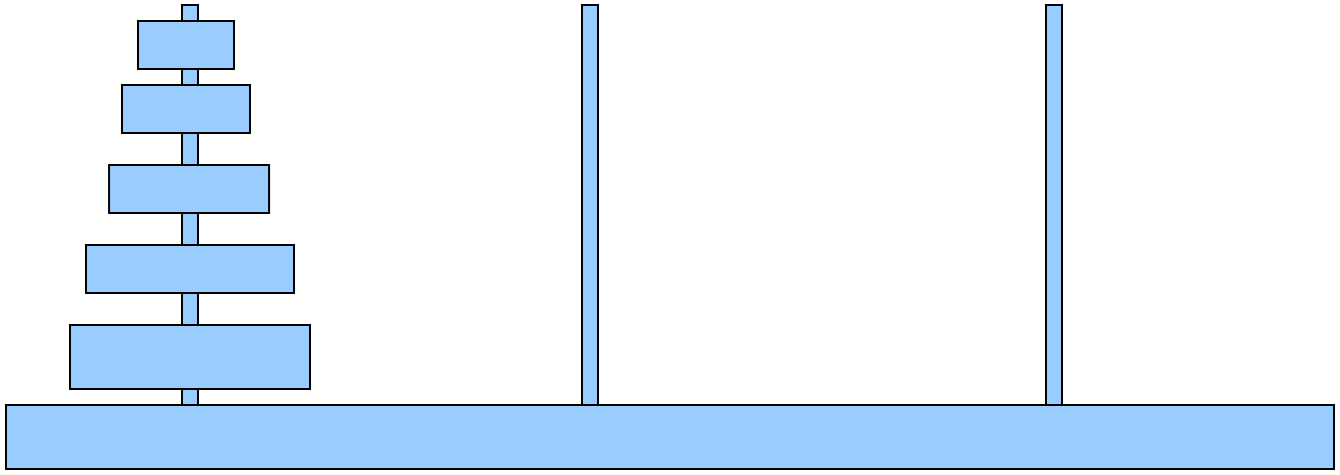
- methods push and pop for LIFO operations
- method peek to obtain top element without removing it from stack
- Example:

```
Stack s = new Stack();
s.push(anObj);
obj=s.pop();
```

A (UML like) view of the above class



The Tower of Brahma (Also called Tower of Hanoi)



Assignment problem

- Implement a stack class
- Use the above stack class to solve the problem of tower of brahma