

# 1999 APS105S Lecture 33: Problems III

Last Day — solved some recursion problems

Today — solve linked list problems

News — tutorial today

- first half: cover some Java useful in some projects
- second half: Q&A... you ask, I answer...

## Projects

- REMINDER: work hard on project this weekend
  - if stuck, try to go on with another part
  - get help from TAs in the lab
- PROJECT DEMO DAY (April 5/6)
  - ready for marking immediately
  - SUBMIT your source code after demo (like the lab test)
  - CHANGE: project report handed in the last class (april 9)

## Objects vs. Static

- question: when do I use static? when do i leave it out
- answer:
  - if you are not making an object
    - always use static
  - else
    - you are making an object
    - use static only to share a value with other objects in the class
  - end if
- question: when do you want to use objects?
- answer:
  - when you want to create more than one of the same thing!

Let's make the deck of cards, shuffle, then print

```
public class Deck
{
    static Card[] deck; // only 1 deck,
    // which is accessed by static main()

    public static void main( String[] args )
    {
        final int numCards = Card.values.length
            * Card.suits.length;

        deck = new Cards[numCards];

        // since deck is an array of objects,
        // initial values of deck[i] == null.
        // each card must be individ. created.
        int s, v, card=0;
        for(s=0; s<Card.suits.length; s++ ) {
            for(v=0; v<Card.values.length; v++) {
                deck[card] = new Card(v,s);
            }
        }

        // shuffle the deck
        final int NUMSHUFFLES = deck.length*100;
        for( int i=0; i < NUMSHUFFLES; i++ ) {
            int c1 = i % deck.length;
            int c2 = (int)(Math.random()*52);
            //swap deck[c1] and deck[c2] (omitted)
        }

        for( int i=0; i<deck.length; i++ ) {
            System.out.println("Card: "+deck[i]);
        }
    }
}
```

## Example

suppose we wish to create a deck of playing cards.

- each card has a face value and a suit:
  - values: 1, 2, 3, ... 9, 10, Jack, Queen, King
  - suits: Hearts, Diamonds, Clubs, Spades
- there is one deck, many cards
- let's make the cards into objects
  - each card has its own value, suit -> instance variables
  - they should all know about range of values, suits -> static

```
class Card
{ public int value; // instance var.
  public int suit; // instance var.

  public Card( int v, int s ) {
    value = v; // check v < values.length ?
    suit = s; // check s < suits.length ?
  }

  // shared by ALL card objects
  public static String[] values = {
    "Ace", "2", "3", "4", "5", "6", "7",
    "8", "9", "10", "Jack", "Queen", "King"
  };

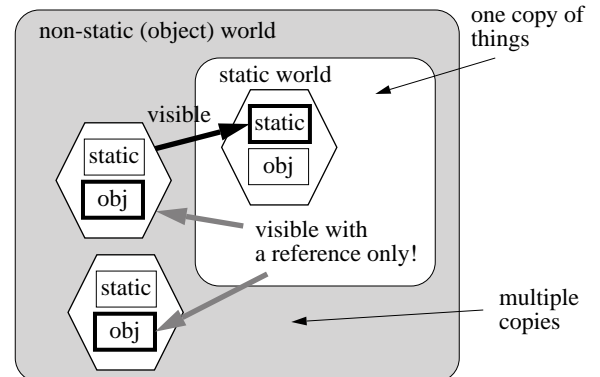
  // shared by ALL card objects
  public static String[] suits = {
    "Hearts", "Diamonds", "Clubs", "Spades"
  };

  public String toString() {
    return values[value] + " of "
      + suits[suit];
  }
}
```

Key notes:

- main is static
  - it accesses a class variable called deck
  - deck must be static because main is static!
- but main can access some objects (non-static things)
  - the cards are not static
  - it can access card toString() methods
  - because it has a REFERENCE to the card object
- cards are objects (not static)
  - but they can contain static things: suits, values arrays
  - these objects share the static things
  - non-static things are individual to each object created

Diagram of the Universe:



What if we wish to deal the deck into a number of hands?

## Arrays vs. Linked Lists

how do you know when to use an array?

when to use a linked list?

### Properties of arrays

- fixed length, not easily resized
- can access any element using [i]
  - -> can do binary search
- not easy to insert new element between 2 adjacent elements
- easy to sort
- not easy to split into 2 arrays

### Properties of linked lists

- variable length, easily resized
- cannot access any element, must go in order, head to tail
  - -> **cannot** do binary search!
- easy to insert new element between 2 adjacent elements
- not as easy to sort
- easy to split into 2 lists
  
- was it wise to use an array for the deck of cards?
- probably, we didn't do much in that program!
- what if we want to do other things?
  - perfect shuffle:
    - split deck in half {0, 1, 2, ..., 26} {27, 28, 29, ..., 51}
    - interleave elements from each half {0, 27, 1, 28, ... 51}
  - random 2-pile shuffle:
    - put top card into left or right pile (random decision)
    - stack piles together
  - deal into hands:
    - take top card from deck, add to hand n++
- some of these things have unpredictable size requirements...

### Example

- use the piles to shuffle the cards

```
public static RandomShuffle
{
    public static void main( String[] args )
    {
        final int numCards = Card.values.length
            * Card.suits.length;

        Pile deck = new Pile();
        // since deck is a reference to an object,
        // we must do a 'new' to create one.

        // we must also create each card
        int s, v;
        for(s=0; s<Card.suits.length; s++) {
            for(v=0; v<Card.values.length; v++) {
                Card c = new Card(v,s);
                deck.insert( c );
            }
        }

        // shuffle the deck into 2 halves
        Pile halfDeckA = new Pile();
        Pile halfDeckB = new Pile();

        while( deck.head != null ) {

            // unlink the top card from the deck
            Card topCard = deck.head;
            deck.head = topCard.next;
            topCard.next = null; // not necessary
```

- different piles of cards
  - each of different (unpredictable) size
  - use a linked list for each pile
- more than one pile
  - make the pile an object
- now we can:
  - insert a new card anywhere in the pile
  - remove one card from the middle of the pile
  - grow or shrink the pile easily

- create a pile of cards!

```
class Card
{
    ... same as before ...
    Card next;
}

public class Pile
{
    // this is a linked list of cards

    public Card head;

    public Pile() { head = null; }

    public void insert( Card c ) {
        c.next = head;
        head = c;
    }

    ... etc as before ...
}
```

Notice

- no use of \*static\*
- we can add special operations to the pile (eg: split, reverse)

```
// link the top card into either pile
double r = Math.random();
if( r < 0.5 )
    halfDeckA.insert( topCard );
else
    halfDeckB.insert( topCard );
}

// merge the two deck halves
if( halfDeckA.head == null ) {
    deck = halfDeckB; // not much shuffled!
} else {
    // find the last card in halfDeckA
    Card lastCard = halfDeckA.head;
    while( lastCard.next != null ) {
        lastCard = lastCard.next;
    }
    // join lastCard of deckA to deckB list
    lastCard.next = halfDeckB.head;
    // assign final reorg'd pile to deck
    deck = halfDeckA;
}

// don't use halfDeckA, halfDeckB anymore

// print the shuffled deck
for(Card c=deck.head; c!=null; c=c.next){
    System.out.println("Card: "+c);
}
}
```