Linked data structures

Johan Montelius

KTH

HT23

a record/object/struct

A data structure with a fixed set of (named) properties.

Properties could be of different types.

```
class Person {
  public String name;
  public Adress adress;
  public int age;

  :
  :
}
```

a record/object/struct

A data structure with a fixed set of (named) properties.
Properties could be of different types.

```
class Person {
  public String name;
  public Adress adress;
  public int age;

  :
  :
}
```

a record/object/struct

A data structure with a fixed set of (named) properties.
Properties could be of different types.

```
class Person {
  public String name;
  public Adress adress;
  public int age;

  :
  :
}
```

a record/object/struct

Objects can be created and their properties used.

```
Person anders = new Person( ...);
  :
String greeting = "Hej " + anders.name;
  :
```

Nothing new, you all know this.

a record/object/struct

Objects can be created and their properties used.

```
Person anders = new Person( ...);
  :
String greeting = "Hej " + anders.name;
  :
```

Nothing new, you all know this.

let's play some cards

```
class Card {
  public Suite suite
  public int value;

  public Card(Suite s, int v) {
    suite = s;
    value = v;
  }
}
```

```
public enum Suite {
    HEART,
    SPADE,
    DIAMOND,
    CLUB
}
```

a deck of cards

```
class Deck {
  Card[] cards;
  first = 0;

  public Deck() {
    cards = Cards[4];
    first = 0;
  }

  public void add(Card crd) {

    :
  }
}
```

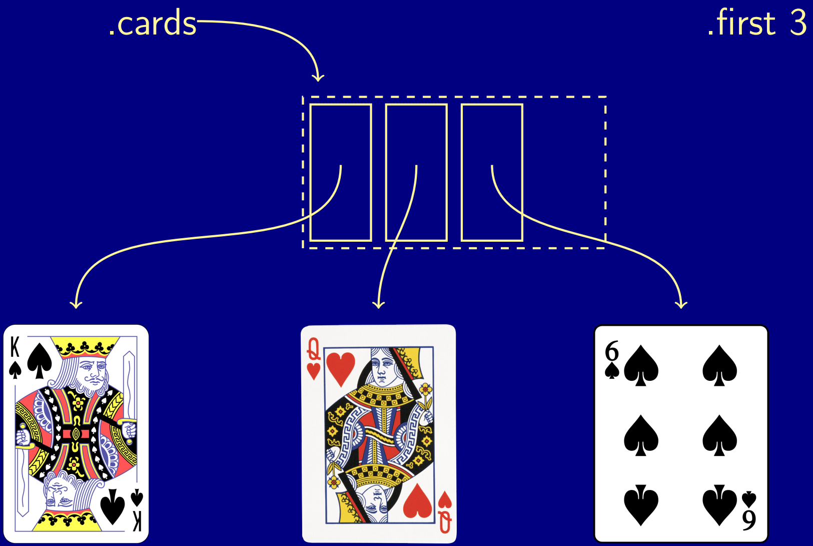
We've seen this before.

a deck of cards

```
class Deck {
  Card[] cards;
  first = 0;

  public Deck() {
    cards = Cards[4];
    first = 0;
  }

  public void add(Card crd) {

    :
  }
}
```

We've seen this before.

a deck of cards



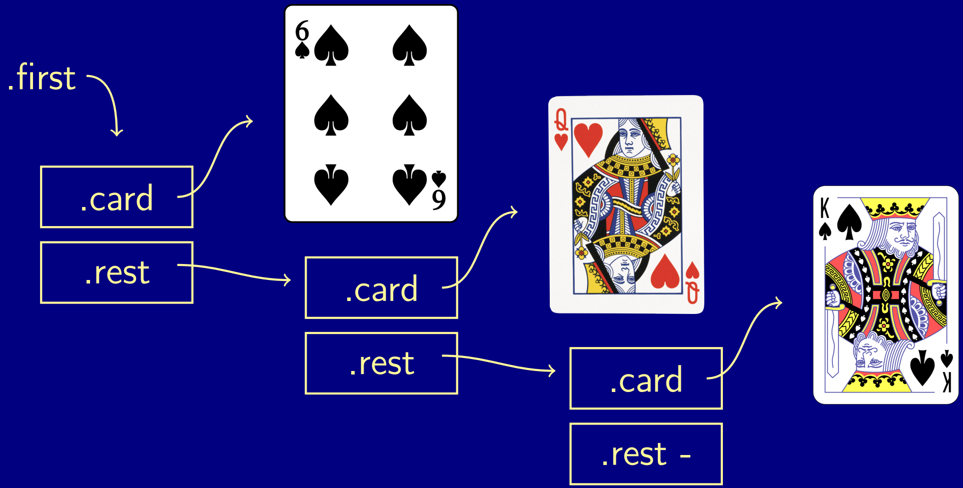
.cards

.first 3

how about this

```
class Deck {
   public Cell first;

   private class Cell {
     Card card;
     Cell rest;
   }

   public Deck() {
     first = null;
   }
   :
```

how about this



.first

.card

.rest

.card

.rest

.card

.rest -

pros and cons

Access the n'th card.

- The list of cards has an $O(n)$ access operation.
- The array of cards has an $O(1)$ access operation.

pros and cons

Access the n'th card.

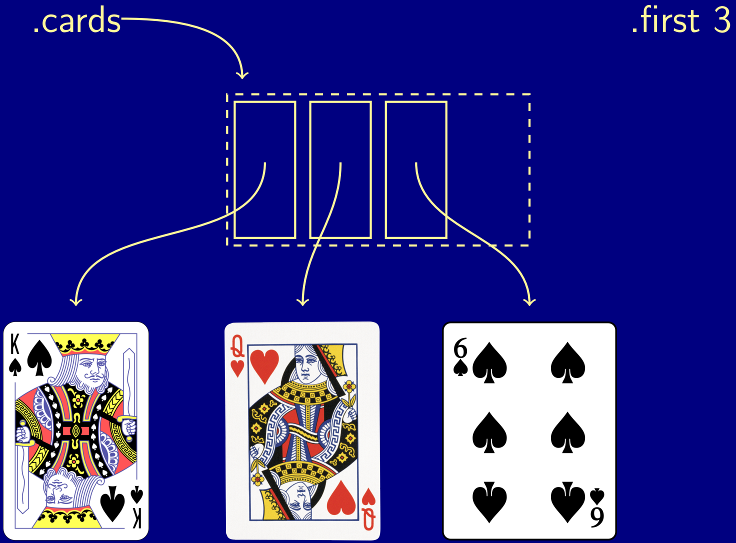
- The list of cards has an $O(n)$ access operation.
- The array of cards has an $O(1)$ access operation.

pros and cons

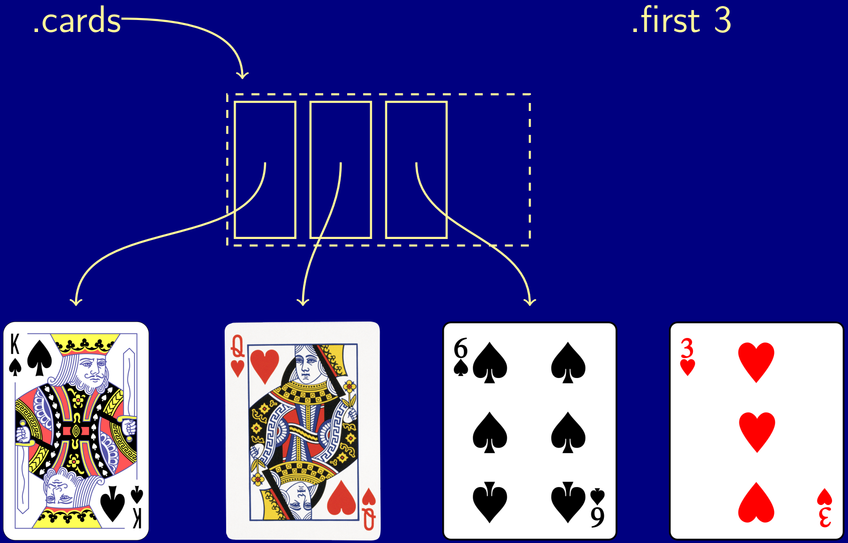
Access the n'th card.

- The list of cards has an $O(n)$ access operation.
- The array of cards has an $O(1)$ access operation.

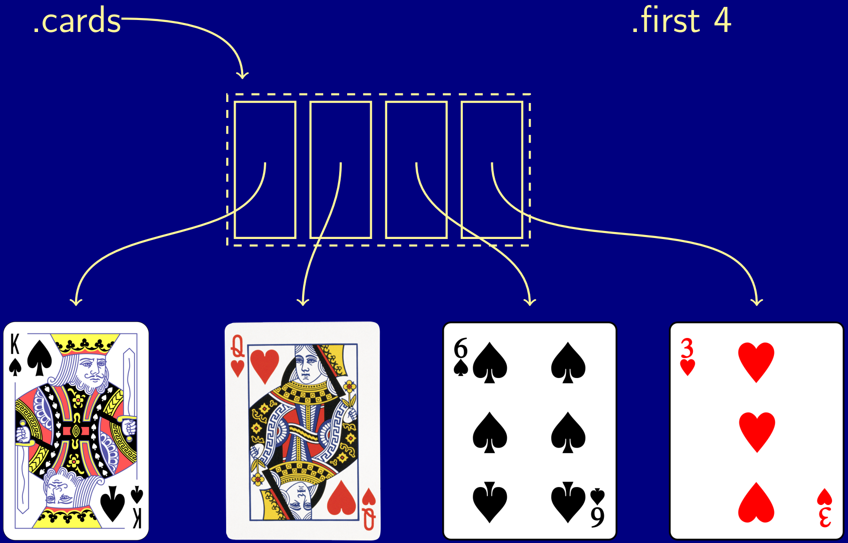
adding a card to an array of cards



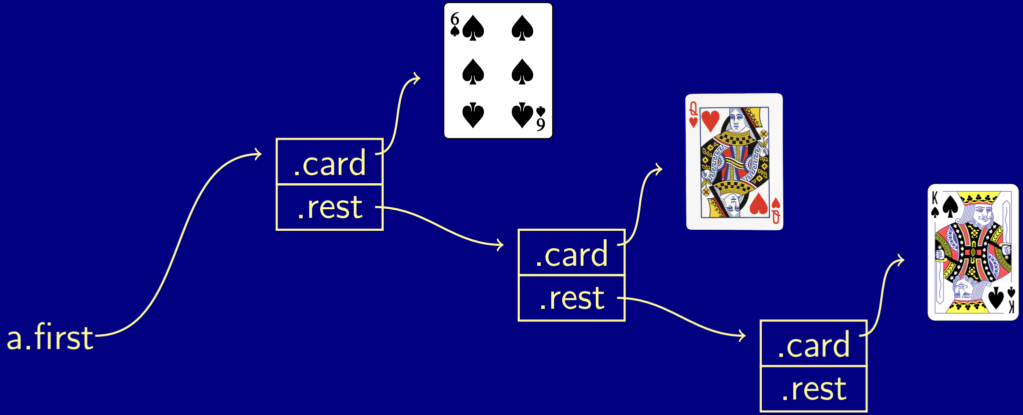
adding a card to an array of cards



adding a card to an array of cards



adding a card to a list of cards



a.first

.card

.rest

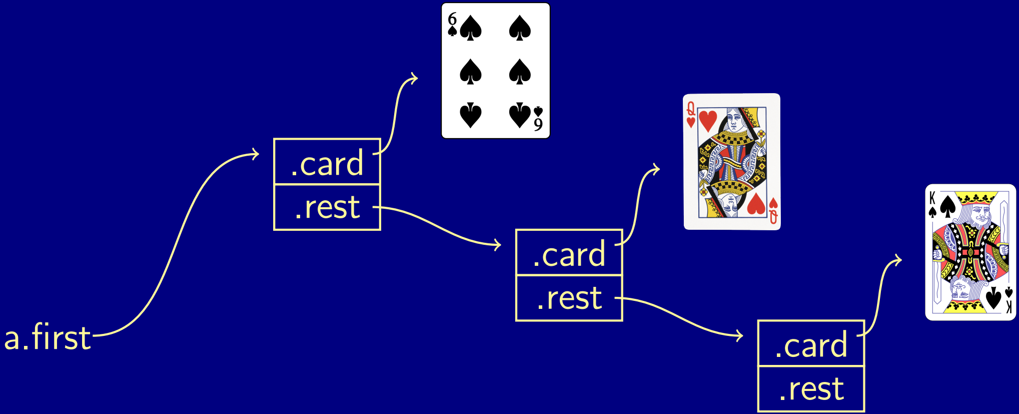
.card

.rest

.card

.rest

adding a card to a list of cards



a.first

.card

.rest

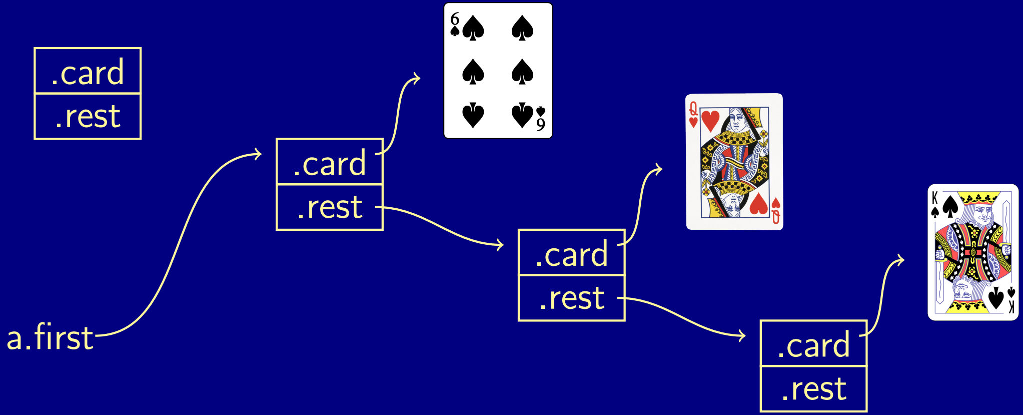
.card

.rest

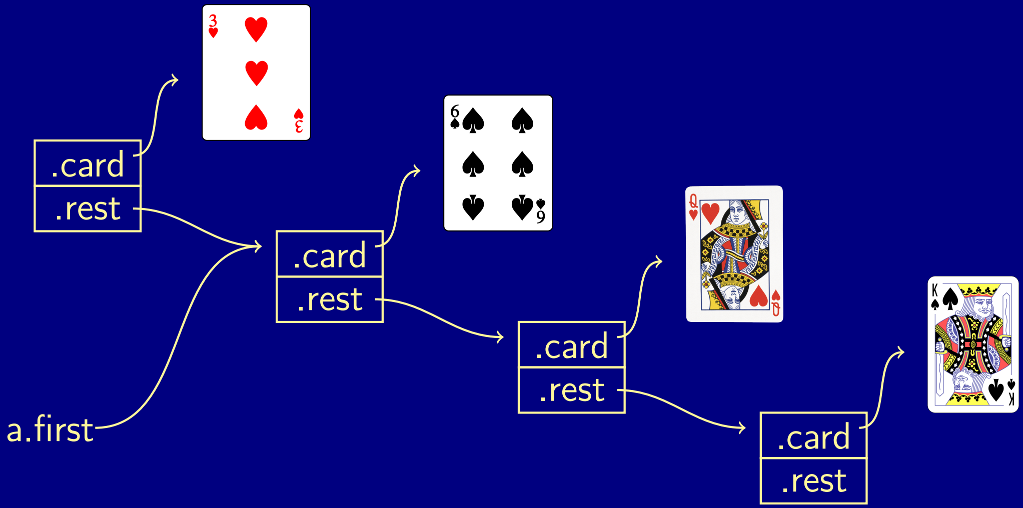
.card

.rest

adding a card to a list of cards



adding a card to a list of cards



.card
.rest

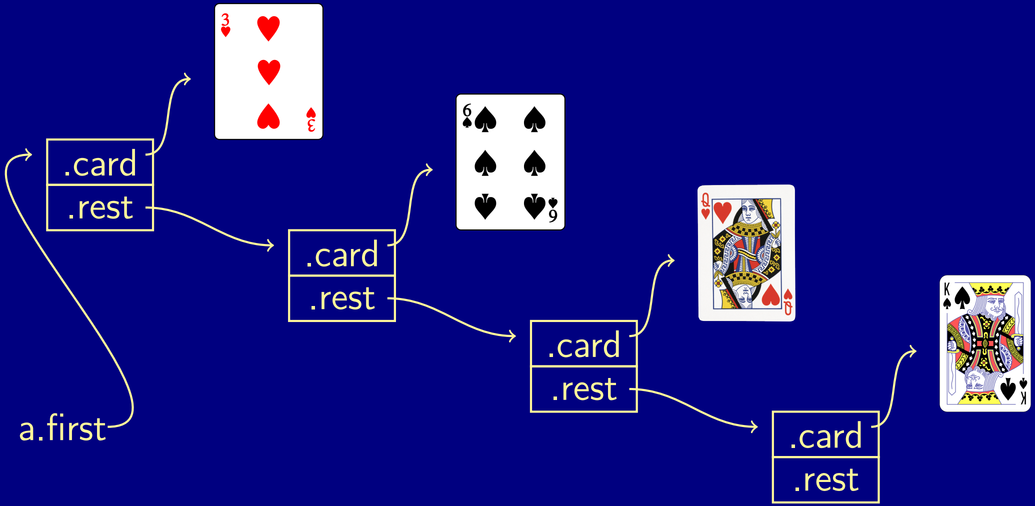
.card
.rest

.card
.rest

.card
.rest

a.first

adding a card to a list of cards



pros and cons

pros and cons

Adding a card has a time complexity of ...

pros and cons

Adding a card has a time complexity of ...

- a list of cards: $O(1)$

pros and cons

Adding a card has a time complexity of ...

- a list of cards: $O(1)$
- a *dynamic array*: amortized cost of $O(1)$

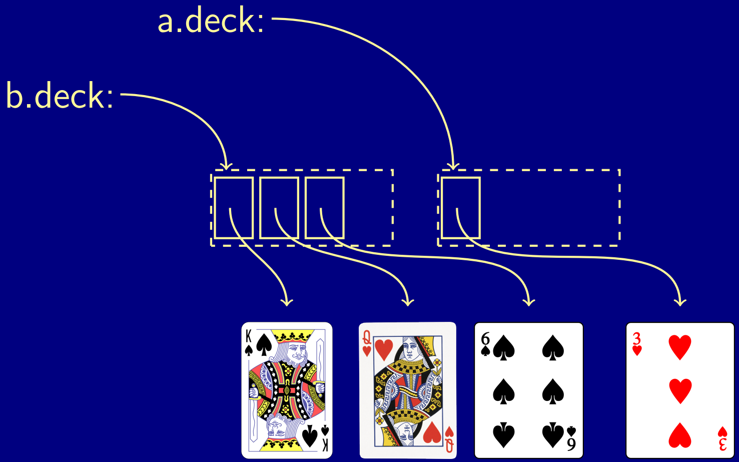
append one deck to another

append one deck to another

Assume we have two decks of cards, `a` and `b`, how do we *append* `b` to `a` i.e. the deck `a` will after the operation hold all cards and `b` should be empty.

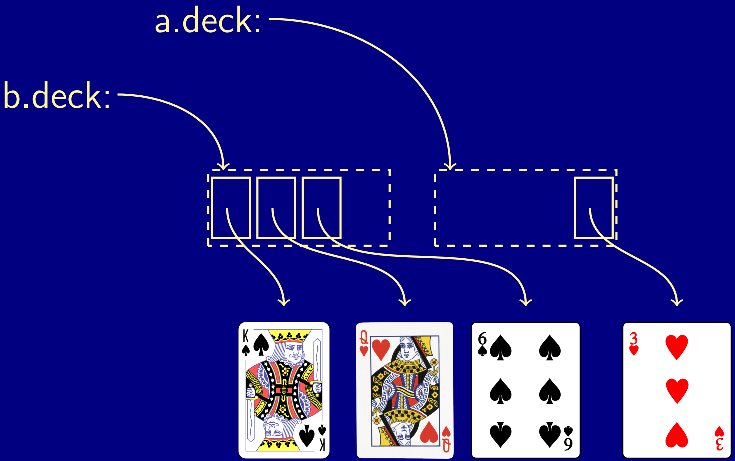
append an array of cards

```
a.append(b);
```



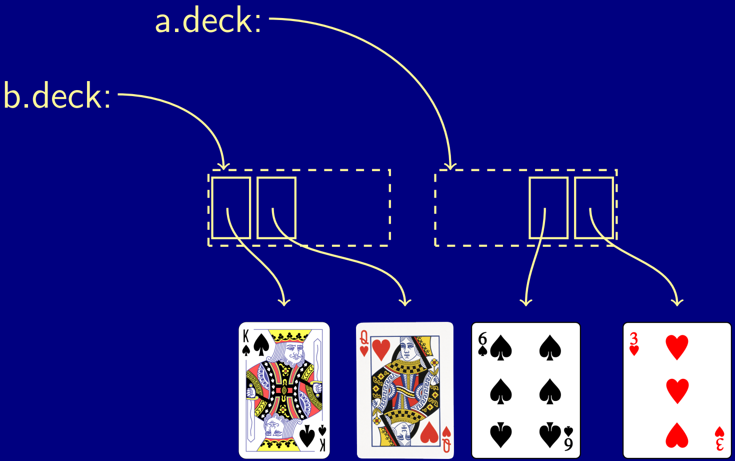
append an array of cards

```
a.append(b);
```



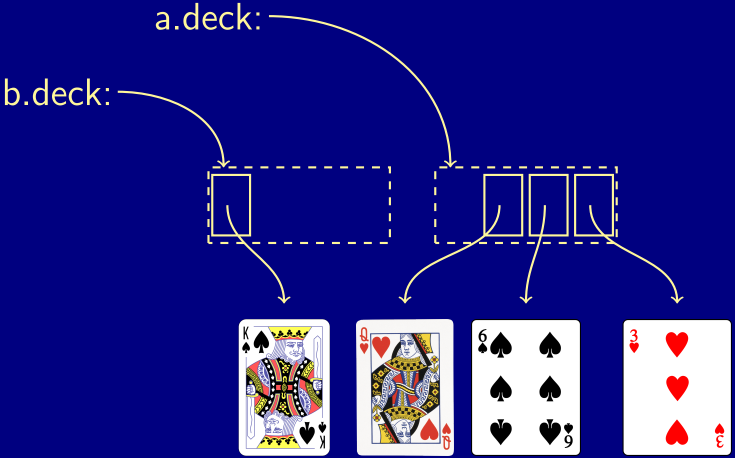
append an array of cards

```
a.append(b);
```



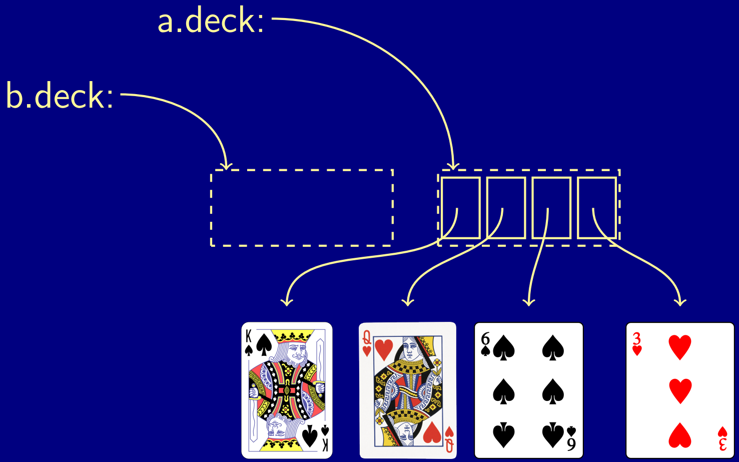
append an array of cards

```
a.append(b);
```



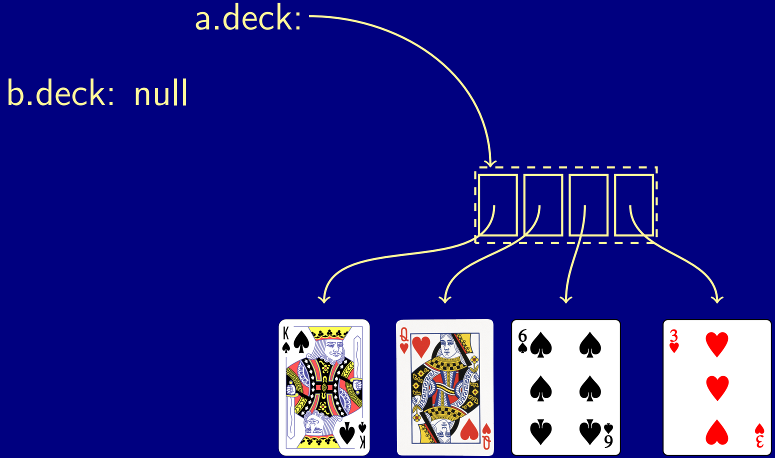
append an array of cards

```
a.append(b);
```



append an array of cards

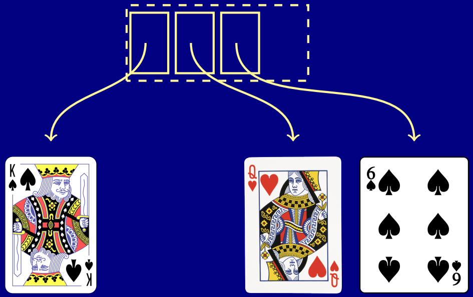
```
a.append(b);
```



append a list of cards

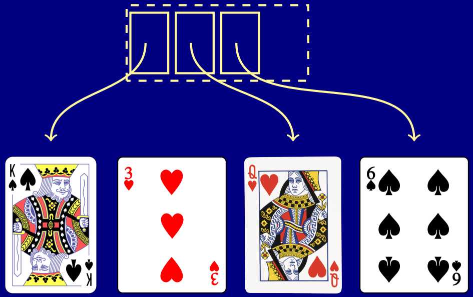
insert a card in an array

a.deck:



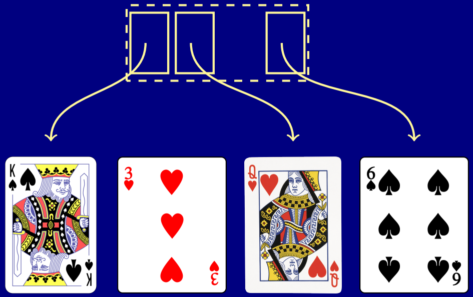
insert a card in an array

a.deck:



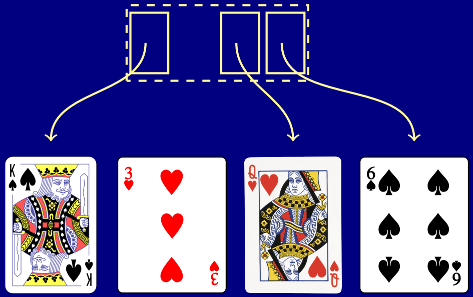
insert a card in an array

a.deck:



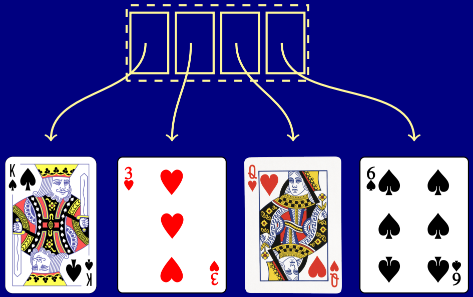
insert a card in an array

a.deck:

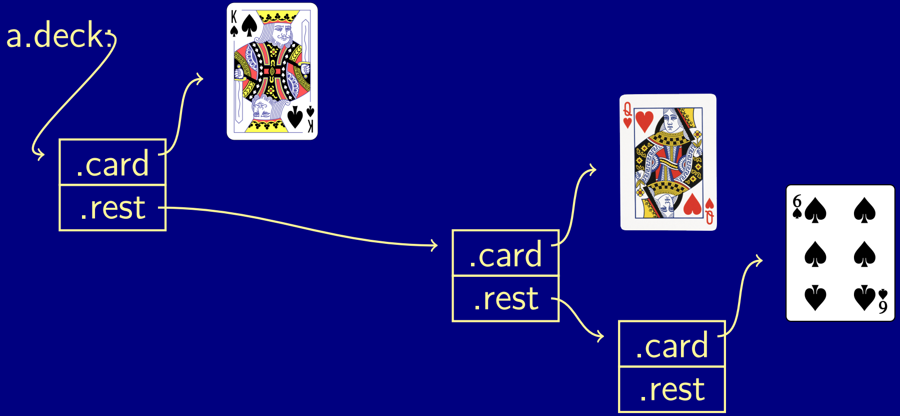


insert a card in an array

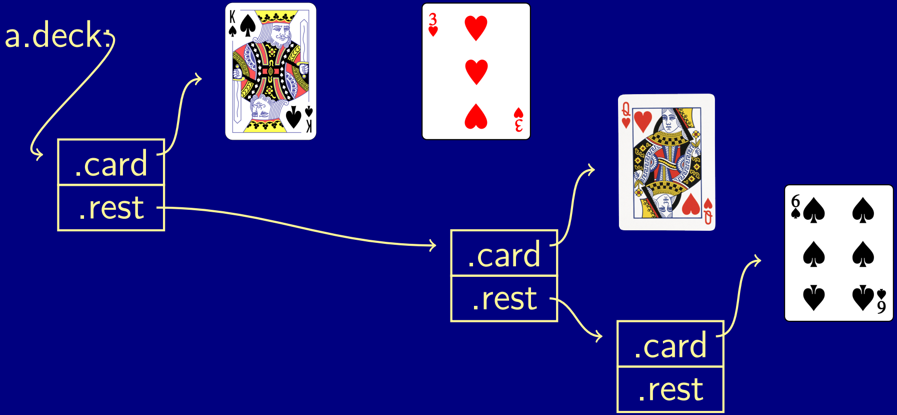
a.deck:



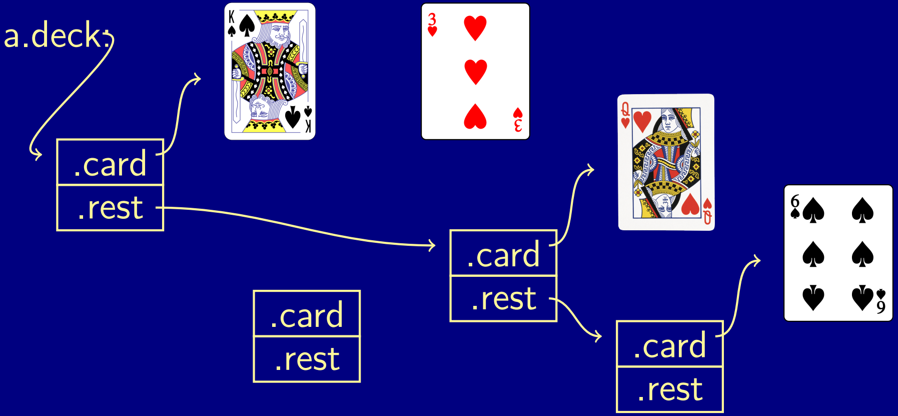
insert a card in a list



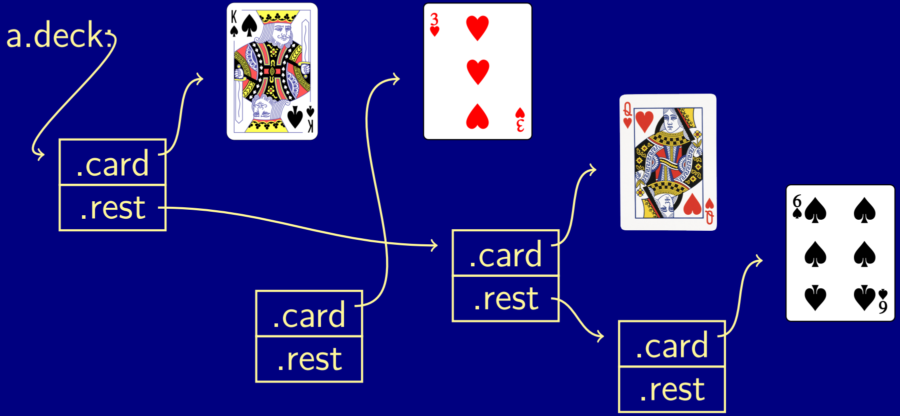
insert a card in a list



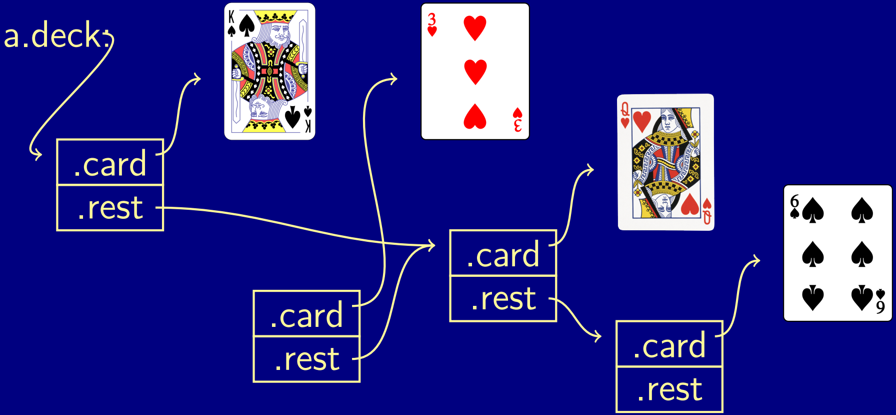
insert a card in a list



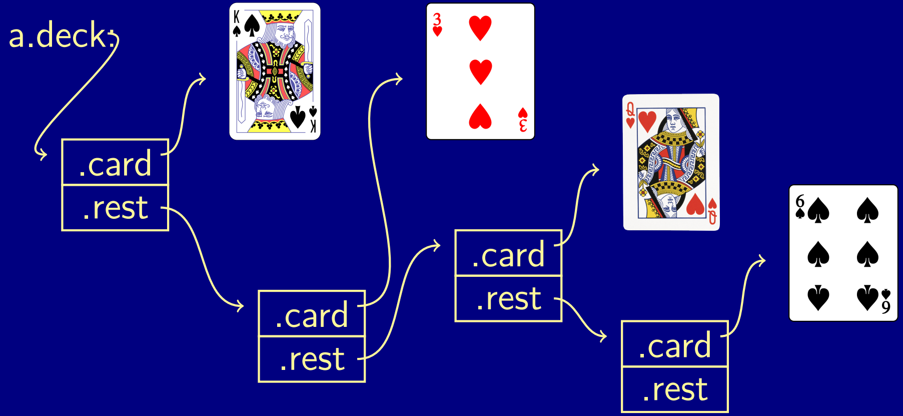
insert a card in a list



insert a card in a list



insert a card in a list



pros and cons

pros and cons

Inserting a card.

pros and cons

Inserting a card.

- The list of cards has an $O(n)$ insert operation....

pros and cons

Inserting a card.

- The list of cards has an $O(n)$ insert operation....
- ..., only $O(n)$ read operations and $O(1)$ write operations.
- The array of cards has an $O(n)$ insert operations ...

pros and cons

Inserting a card.

- The list of cards has an $O(n)$ insert operation....
- ..., only $O(n)$ read operations and $O(1)$ write operations.
- The array of cards has an $O(n)$ insert operations ...
- ..., $O(n)$ read and write operations.

LinkedList

```
class LinkedLints {
  Cell first;

  private class Cell {
    int head;
    Cell tail;
     :
  }

  public LinkedList() {
    first = null;
  }
  :
}
```

LinkedList

```
class LinkedLints {
  Cell first;

  private class Cell {
    int head;
    Cell tail;
     :
  }

  public LinkedList() {
    first = null;
  }
  :
}
```

The `Cell` data structure is also referedd to as a *cons cell*.

LinkedList - search

```
public boolean search(int key) {
  Cell nxt = first;
  while (nxt != null) {
    if (nxt.head == key)
      return true;
    nxt = nxt.tail;
  }
  return false;
}
```

LinkeList - what?

```
public void what(int key) {
  Cell nxt = first;
  Cell prv = null;
  while (nxt != null) {
    if (nxt.head == key) {
      if (prv != null)
        prev.tail = nxt.tail;
      else
        first = nxt.tail;
      return;
    }
    prev = nxt;
    nxt = nxt.tail;
  }
  return;
}
```

LinkedList - append

```
public void append(LinkedList b) {
  Cell nxt = first;
  while (nxt.tail != null) {
    nxt = nxt.tail;
  }
  nxt.tail = b.first;
  b.first = null;
}
```

LinkedList - append

```
public void append(LinkedList b) {
  Cell nxt = first;
  while (nxt.tail != null) {
    nxt = nxt.tail;
  }
  nxt.tail = b.first;
  b.first = null;
}
```

There is an error in this code - find it.

Stack

```
class Stack {
  Cell stack;

  public void Stack() {
    stack = null;
  }
  :
  :
}
```

Stack - push n pop

```
public void push(int item) {
  stack = new Cell(item, stack);
}
```

Stack - push n pop

```
public void push(int item) {
  stack = new Cell(item, stack);
}

public int pop() {
   if (stack == null) {
     throw new Exception("pop from empty stack");
   }
   int ret = stack.head;
   stack = stack.tail;
   return ret;
}
```

linked lists

linked lists

- $O(n)$ to find the right position

linked lists

- $O(n)$ to find the right position
- $O(1)$ to perform operation once position is found

linked lists

- $O(n)$ to find the right position
- $O(1)$ to perform operation once position is found
- often simple to work with
- a dynamic stack