Class design

From last time: finish Names example

(from last Thur lect slides)

- practice with
  - coding array algorithms
  - implementing classes
  - and using good development techniques
- incremental development
- for lookup, **remove**, insert:
  - design test cases first
  - implement code
    - code refactoring
  - test code

Today's Class Design topics

- Preconditions
- Class invariants
  - representation invariants
  - testing repr. invariants

Announcements

- This week's lab: milestone for PA2 (today's lecture helpful for lab)
- Don't wait until after MT to start PA2
- Midterm 1 is on Tue 2/23 9:30am – 10:50am
  - 6:30 – 7:50am for students in Asia
  - email coming soon confirming your designated exam time
  - sample problems have been published
  - Closed book, closed note, no outside s/w, websites, or electronic devices
  - blank scratch paper ok for scratch work (not submitted)
  - online exam
  - rehearsal exam coming soon

Method preconditions

- a restriction on how a method can be called
  - Ex (from book): in **BankAccount** class

    void deposit(double amount)

    Precondition:

- document any preconditions in the method comment

- why not

  "amount must be type double" ?

Method contract

- client must satisfy precondition
- a contract between client code and method:
  - if you call the function this way,
    we guarantee it will do what we say it does
  - otherwise, behavior is undefined
- avoid performing duplicate checks between client and method code

POLL: preconditions

- **BankAccount** example

  void deposit(double amount)

  Precondition: **amount > 0**

Asynchronous participation: Link to Preconditions poll

What should method do?

- a call that violates the precond is incorrect (remember: undefined results)

- Java **assert** statement is useful:

  assert amount > 0;

Restrictions on implicit parameter

`x.foo();`

Another reason for a precond:

- restriction on *when* certain methods can be called
  - object can be in different states
- Illegal to call `next()` when `Scanner` has no more input (eof in lab4)
- **PRE: hasNext() is true**
- Try to minimize them

Your Precondition comments

- Two ways to document at the top of a method:
- Javadoc style (next to param in question):

```
@param amount
        the amount of money to deposit,
        must be > 0
```

- Or state all preconditions on separate line:

```
PRE: amount > 0
```

Class Invariants

- a statement about an object that's always true between method calls:
  - true after constructor
  - true after every mutator
  - (therefore, also true before every method call)
- interface invariant: true from client view
- representation invariant: true about object representation

Interface Invariants

- sometimes related to preconditions
- Example in book: **BankAccount**
  Invariant: getBalance() >= 0
- would document in overall class comment
- For **CoinTossSimulator** class:
  Invariant: getNumTrials() =
  **   getTwoHeads() + getTwoTails() +**
  **   getHeadTails()**
- For **Names** class
  Invariant: names are in alphabetical order
  **   and are unique**

Representation invariants

- a statement about the *internal object representation* that's always true between method calls:
  - true after constructor
  - true after every mutator
  - (therefore, also true before every method call)
- describes valid internal state of the object
  - any restrictions on what can be in instance variables
  - any relationships between values in different instance variables

Ex: Repr. invar. for **Names** class

- … that uses ***ArrayList*** representation

```
class Names {
    . . .
    private ArrayList<String> namesArr;
    /* Representation invariant:
        -- names are unique
        -- names are in alphabetical order in namesArr
        -- number of names stored is namesArr.size()
    */
}
```

Ex 2: Repr. invariant for **Names** class

- … that uses ***partially filled array*** representation

```
class Names {
  . . .
  private String[] namesArr;
  private int numNames;
}
```

Names						
	0	1	2	3	4	· · ·
namesArr	Ann	Bob	Jun			· · ·
numNames	3					
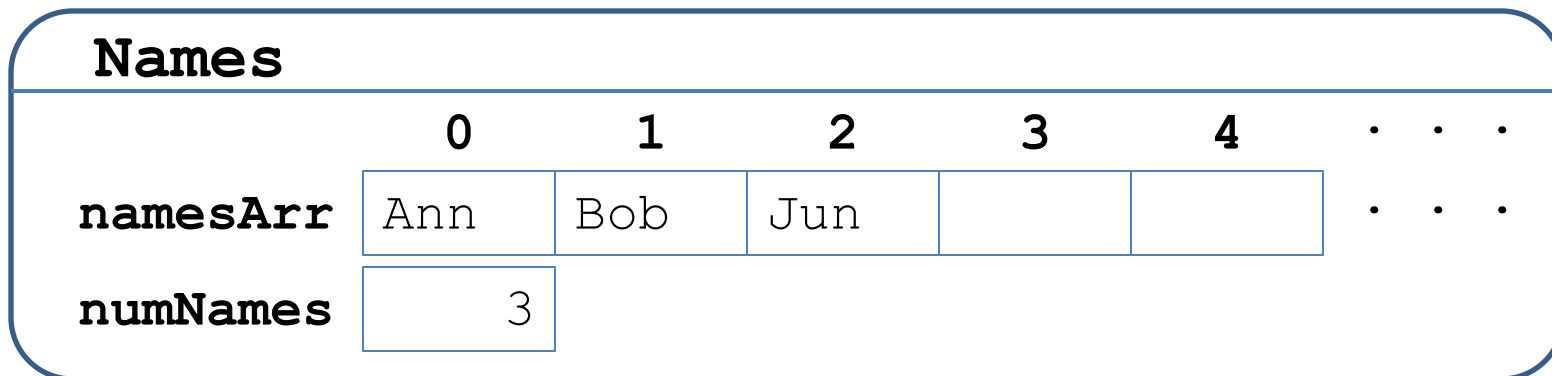
Ex 2 of repr. invariants (cont.)

Names							
	0	**1**	**2**	**3**	**4**	· · ·
namesArr	Ann	Bob	Jun			· · ·
numNames	3					

repr. invariant:

- **numNames is the number of names**

- **0 <= numNames <= namesArr.length**

- **if numNames > 0, the names are in namesArr
        locs: [0,numNames)**

- **names are in alphabetical order**

- **names are unique**

Different invar. with same data types

```
class Names {
   . . .
   private String[] namesArr;
   private int lastLoc;
}
```

Names						
	0	1	2	3	4	· · ·
namesArr	Ann	Bob	Jun			· · ·
lastLoc	2					

Different invariant (cont.)

	0	1	2	3	4	· · ·
namesArr	Ann	Bob	Jun			· · ·

Names

lastLoc 2

- representation invariant:

Testing representation invariants

- Can use **assert** for sanity checks.
- One kind of sanity check:

  check representation invariant
- Write a *private* method:

  boolean isValidObject()
- at end of every method:

  assert isValidObject();
- You will be doing this in pa2.